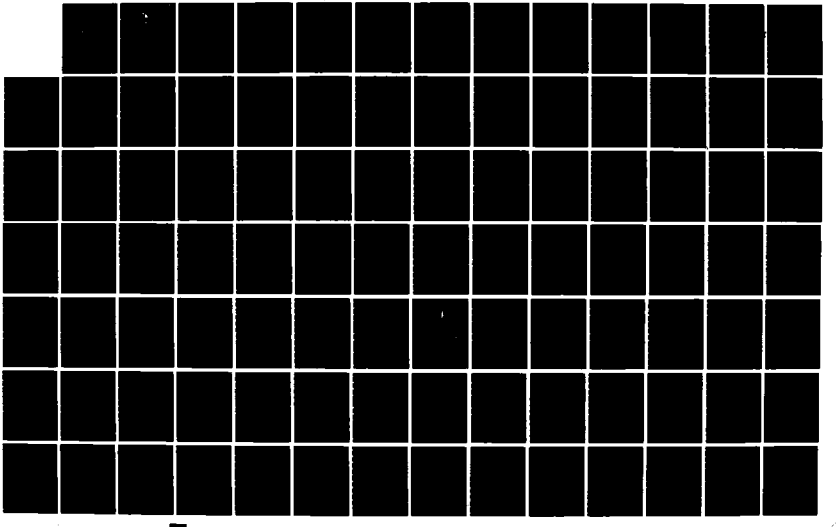
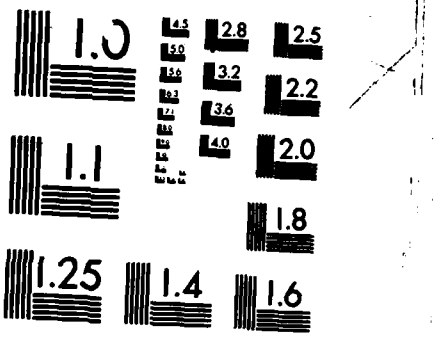
AD-A147 552 A SURVEY AND EVALUATION OF SOFTWARE QUALITY ASSURANCE 1/2
(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH
SCHOOL OF SYSTEMS AND LOGISTICS S P LAMB SEP 84
UNCLASSIFIED AFIT/GSM/LSY/84S-19 F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0


# A SURVEY AND EVALUATION OF SOFTWARE QUALITY ASSURANCE

THESIS

Steven P. Lamb
Captain, USAF

AFIT/GSM/LSY/84S-19



DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

2

AFIT/GSM/LSY/84

# A SURVEY AND EVALUATION OF SOFTWARE QUALITY ASSURANCE

THESIS

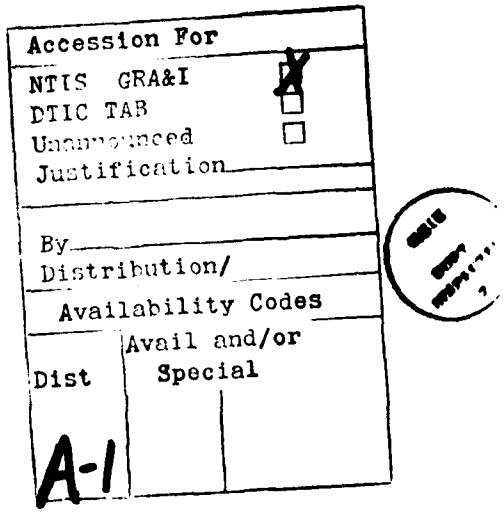
Steven P. Lamb
Captain, USAF

AFIT/GSM/LSY/84S-19



Approved for public release; distribution unlimited

The contents of the document are technically accurate, and no sensitive items, detrimental ideas, or deleterious information are contained therein. Furthermore, the views expressed in the document are those of the authors and do not necessarily reflect the views of the School of Systems and Logistics, the Air University, the United States Air Force, or the Department of Defense.



AFIT/GSM/LSY/84S-19

# A SURVEY AND EVALUATION OF SOFTWARE QUALITY ASSURANCE

THESIS

Presented to the Faculty of the School of Systems and Logistics
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Systems Management

Steven P. Lamb, B.S.
Captain, USAF

September 1984

## Acknowledgements

I would like to take this opportunity to express my appreciation to all the many people who helped make this thesis possible. I am particularly grateful for the guidance and advice of Major Ronald H. Rasch, who helped me focus the thesis research. Also, my sincerest thanks to Mr. Michael D. Bates for his guidance, assistance, and encouragement throughout this thesis effort. Their willing attitudes, timely suggestions, and genuine interests and concerns made this thesis a special learning experience.

Thanks goes to the professionals who agreed to be interviewed. Without their cooperation, this thesis would have been impossible. Unfortunately, their names cannot be listed as they are too numerous and the subject matter is of such a nature that it was agreed that no names would be used.

Special thanks are extended to Lieutenant Colonel Brian S. Maass, Major Charles E. Beck, and Dr. Charles R. Fenno for their support and interest in this project. Also, special recognition goes to my fellow students for their moral support.

Finally, and most importantly, I wish to thank my mother and father. As always, because of their love and support, I was able to overcome many difficulties during this period in my life.

Steven P. Lamb

## Table of Contents

## List of Figures

## List of Tables

## Abstract

It is crucial to the success of mission critical computer resources (MCCR) that software be delivered for operational use with the minimum number of errors possible. For this reason, the discipline of software quality assurance is needed. This ~~research~~ thesis focuses on data collection by means of an extensive literature review and personal interviews with civilian and Air Force software development organizations. Then analysis was performed to determine what approaches would improve the quality of software before delivery to the Air Force for operational use.

To provide the highest level of software quality, the entire development process must include quality checks at each step from design through acceptance test. An active software quality assurance program that identifies and corrects errors during the development process is necessary. This effort will lead to significant defects being identified and resolved early. If the quality of software is to improve, greater emphasis must be placed on software quality assurance as a separate discipline. Quality software cannot be attained by following hardware oriented plans and procedures. Therefore, software conformance standards must be provided. Technology is constantly changing and advancing, and provision must be

made to update personnel in the state-of-the-art quality assurance practices. Continual training is essential both for those personnel who have quality assurance background and those who do not. The arguments for software quality assurance are critical. In short, they are to combat error and improve software quality to meet mission needs.

# A SURVEY AND EVALUATION OF SOFTWARE QUALITY ASSURANCE

## I. Introduction

Historically, computer technology has been widely exploited in the Air Force, both in command and control systems and in management support systems. In addition, this technology has become an essential subsystem of modern aircraft, with regard not only to both offensive and defensive weapon delivery, but also to navigation, to on-board monitoring of performance sensors, to automated ground support equipment, and even to the actual flight control of the vehicle (24:1). Therefore, computer hardware failure or inadequate software can mean that an aircraft cannot fly or carry out its mission, command and control systems cannot communicate with each other, or important management information is not received in a timely or accurate manner. Given these problems, it becomes important to control the computer, and insure the quality of computer hardware and software.

### Terms Defined

Hardware refers to any and all physical machines forming a computer (14:102). This would include card readers, printers, tape units, disk drives, central processing units, consoles, terminals, and so on.

Software is defined as a combination of computer programs and computer data required to enable the computer hardware to perform computational or control functions (5:98). Software can be expressed in human readable form, such as a source listing or other documentation. It may reside in media only accessed by the computer, such as magnetic tapes or disks. Also, it can reside within the computer memory (5:99).

Quality is, at best, a relative and subjective measurement. Webster's New Collegiate Dictionary defines quality as a distinguishing attribute or characteristic (54:936). Defining these attributes or characteristics becomes a problem because individuals interpret quality from their own perspective.

For the purpose of this research effort, software quality is defined as the degree to which a software product possesses a specified set of attributes necessary to fulfill a stated purpose (5:99).

A current trend is that software quality and software reliability are synonymous. Since quality is a set of many attributes and reliability is one of these attributes, then reliability becomes a subset of quality (40:129). By definition, reliability is the ability of a software program to perform a required function under stated conditions for a stated period of time (34:32).

There are a host of definitions that apply to quality assurance. The official Air Force position is that quality

assurance is a "planned and systematic pattern of all actions necessary to provide adequate confidence that adequate technical requirements are established; products and services conform to established technical requirements; and satisfactory performance is achieved" (18:7). This definition goes beyond earlier interpretations that stated quality assurance should merely verify conformance to specifications. Another definition proposed by the Institute of Electrical and Electronic Engineers states that quality assurance is a "planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical requirements" (34:28). The bottom line for software or hardware quality assurance is ensuring the user's needs have been adequately satisfied.

Quality control is a management function whereby control of quality of produced material is exercised for the purpose of preventing production of defective material (20:5). This differs from quality assurance because "a management function" is only one of "all actions" necessary to provide quality. Therefore, quality control is not as thorough as quality assurance when discussing software; but it is an essential part of quality assurance.

The software life cycle consists of a set of activities occurring in a given order during the development and use of software (34:37). The time periods during which these activities occur are called phases. The software life cycle typically includes a requirements phase, design phase, code

phase, test phase, acceptance phase, and operational phase (9:40). At the current time a consensus has not developed as to which phases comprise a software life cycle. Therefore, the above example is only one of many that exist in today's literature.

## Problem Statement

At present, it is not generally possible to develop computer software that is 100 percent reliable or error-free (25:263). Therefore, when placed into operation, any undetected or uncorrected software errors may result in severe operational problems and possible mission degradation. Software quality assurance is a way to improve quality through reduced defects (41:356). This investment in early error detection reduces later error discovery rates, saves the much larger rework cost, and improves the Air Force mission capability.

## Background

In the U.S., the annual cost of software in 1980 was approximately $40 billion, or about two percent of the Gross National Product (9:17). Compared to the cost of computer hardware, the cost of software is continuing to increase, as shown in Figure 1.

Within the federal government, Dr. Jacques S. Gansler, former Deputy Assistant Secretary of Defense for Material Acquisition, also supports the relevance of software costs.

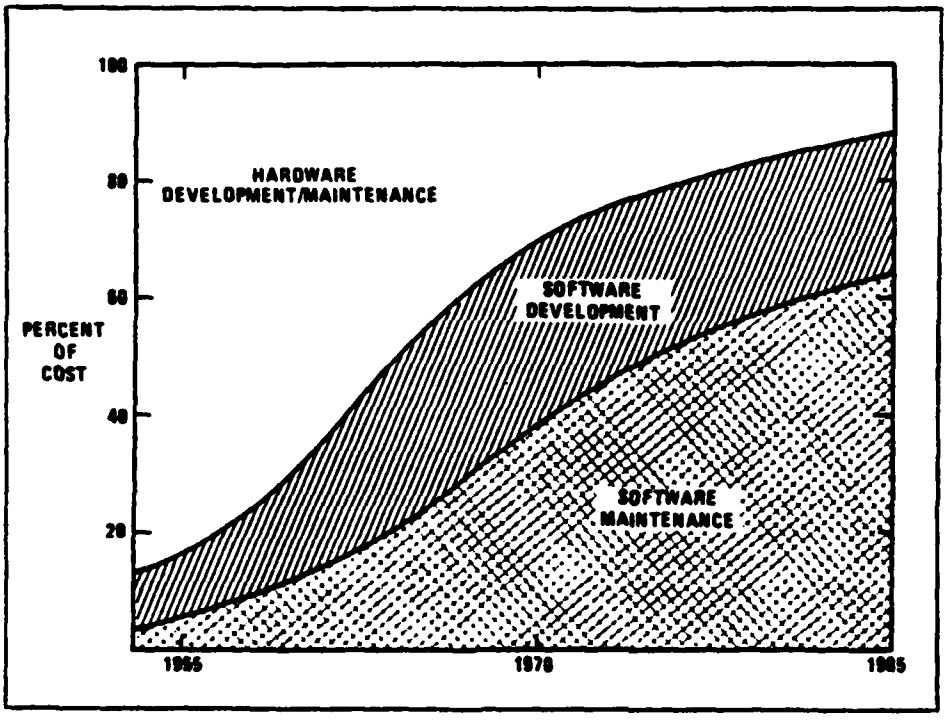



Figure 1. Hardware/Software Cost Trends (13:74)

•

> Software is big business within the Department of Defense. The current annual expenditure on Defense System software is now estimated in excess of three billion dollars; yet even this substantial sum is the tip of the iceberg (28:2).

In addition, a report to Congress by the Comptroller General, General Accounting Office (GAO), FGMSD-80-4, November 9, 1979, cites a continuing problem of developing software with the federal government. The report reflected the views of 163 software contracting firms and 113 federal government project officers. The following is a summarized account:

1. Cost overruns are common in more than 50 percent of the cases studied.

2. Schedule overruns occur in more than 60 percent of the cases.
3. Of the nine contracts examined and the $6.8 million expended, the results were:
    a. Software delivered but never used: $3.2 million.
    b. Software paid for, but never delivered: $1.95 million.
    c. Software extensively reworked before use: $1.3 million.
    d. Software used after changes: $198,000.
    e. Software used as delivered: $119,000.

The GAO report concluded that "the government got for its money less than two percent of the total value of the contracts" (27:77-78).

Software cost is only one facet of the picture. Another facet is unsatisfactory software performance due to software errors. The need for error-free software in major weapon systems is obvious, as stated earlier. But the impact of "when" defects are discovered in the software life cycle needs further discussion.

Defects in software are of two kinds. The first is a design error, an inconsistency with design or specifications which causes the software to do other than what is desired by the user. The second defect is a logic error. This type of error is in the computer program logic that cause the software to operate inconsistently with respect to the written requirements (27:22).

Software error studies have reported design errors as occurring most often, ranging from 61 percent to 64 percent. Logic errors were reported to range from 36 percent to 39 percent (29:13). The important point to note is the large percentage of design errors.

Other studies show that software errors typically are not detected until late in the software life cycle. A study of large software development effort found that 54 percent of all software errors were not discovered until the acceptance phase. The overwhelming proportion of these were design errors (1:352). When software errors are found late, requirements have to be revalidated, designs redesigned, software and systems retested, and documentation rewritten (13:76).

The cost of fixing an error rises dramatically as the software progresses through the life cycle. Figure 2 shows a summary of experience on projects at International Business Machines (IBM), General Telecommunications Equipment (GTE), the Safeguard software project, and several TRW projects. The solid line in this figure represents the relative cost of correcting software errors as a function of the life cycle phase in which the corrections are made (9:39-40). Comparing extreme ends of the software life cycle in Figure 2, one man hour spent in finding and correcting a requirements specification error during the requirements phase would be multiplied by 100 if the same error were discovered and corrected during operations (38:488). Therefore, discovering errors early in the software life cycle can yield large payoffs.

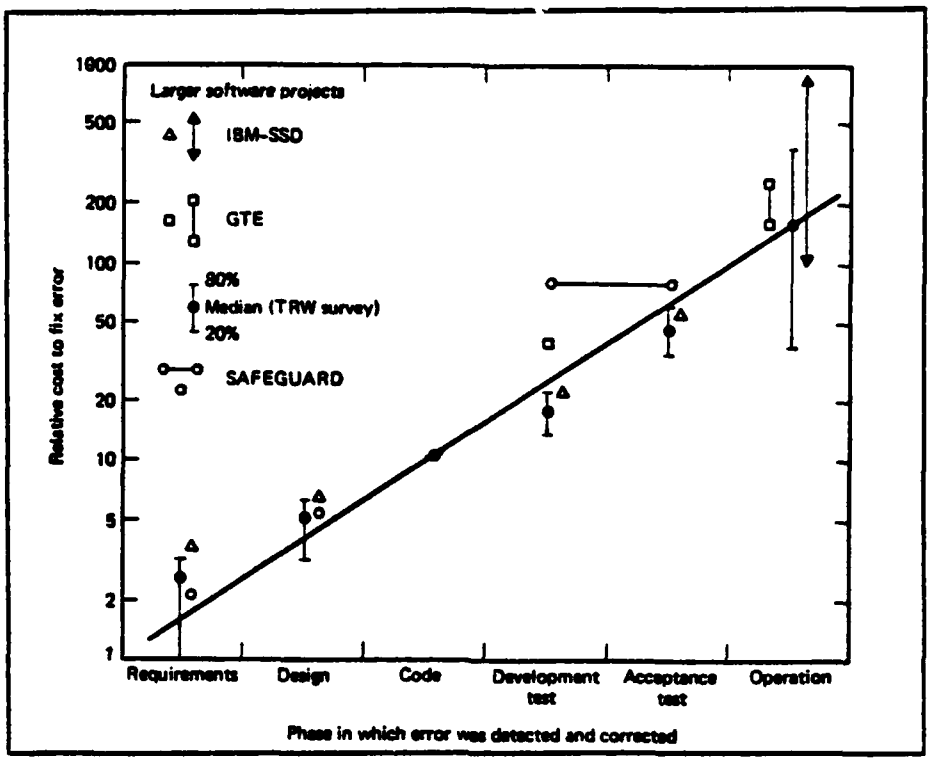



Figure 2. Cost of Fixing Errors (9:40)

This rising software error cost is further supported by a DoD study that reported costs for Air Force avionics software averaged about $75 per instruction during the development phases while the cost of software maintenance during the operational phase ranged $4000 per instruction. Note here that software maintenance, in addition to correction of errors, includes the updating and revision of software caused by changes to the mission requirements (22:311). In addition, J. H. Frame of IBM, in a talk entitled "Major Trends in Software Quality: 1978-1985," said that correcting errors is 38 times more expensive during the test phase than during the initial design effort (27:28).

Software quality assurance not only strives to prevent faults from getting into the software, it also wants to find and correct them as early as possible in the development cycle.

## Research Question

What approaches and/or techniques will improve the quality of computer software before turnover to the Air Force?

## Research Objectives

1. Identify various methodologies used to ensure software quality.
2. Compare methods used by civilian and Air Force organizations to ensure software quality.
3. Critique the effectiveness of the methods used.

## Scope of Research

Individual studies may be found on Software Quality Metrics, Software Control During Acquisition, Software Documentation, and Microcomputer Software; but there is a noticeable lack of studies concerning Software Quality Assurance as the main subject. This thesis will increase the available knowledge on Software Quality Assurance.

Because computer software encompasses a broad subject matter, only the software associated with mission critical computer resources (MCCR) is discussed in this research effort. In general, mission critical computer resources are involved with:

1. Intelligence activities;
2. Cryptologic activities related to national security;
3. Command and control of military forces;
4. Equipment that is an integral part of a weapon or weapon system; or
5. Is critical to direct fulfillment of military or intelligence missions (31:9).

Hardware issues are not within the scope of this research. Hardware, perhaps because it is easier to measure, is constantly reported upon in trade journals (27:1). Therefore, any references beyond simple comparisons are not included.

Civilian and Air Force organizations used in this research were chosen from Dayton, Ohio and Wright-Patterson Air Force Base, Ohio. This was done because there is an extensive industrial base dedicated to software development in this geographical area.

The research effort was also limited to the time period allowed by the Air Force Institute of Technology 15 month graduate degree program.

## II. Software Quality Assurance

The information contained in this chapter is the result of an extensive literature review on software quality assurance. It is presented to increase the reader's understanding of software quality assurance and provide a common foundation for further research and discussion.

To begin with, software quality assurance is a relatively new concept that has received a great deal of emphasis over the past few years. Industry and the government have begun to realize a need for greater discipline in the software development process (30:18).

### Why the Need for Quality Assurance?

The arguments for quality assurance have been given earlier. In short, they are to combat error, thus reducing life cycle cost, and improve software quality to meet mission needs.

Another case for quality assurance is the social motive. Computers and software are making an impact on personal lives. Everyday, more and more personal records, bank accounts, community services, traffic control, air travel, medical services, and national security are being entrusted to computers and software (9:19). This increasing impact on human welfare presents a challenge to software quality assurance.

## Objectives of Quality Assurance

As stated in Webster's New Collegiate Dictionary, an objective is defined as "something toward which effort is directed" (54:785). Therefore, the effort of quality assurance is directed towards the following objectives:

1. Incorporation of software quality assurance into the overall software program planning.
2. Preparation and evaluation of standards that guide the preparation of software documentation, design, and code.
3. Evaluation of the software design process and design products for conformance to requirements.
4. Monitoring of the software design for compliance with design and performance requirements, adequacy of methods used, and positive evidence of compliance.
5. Review of software tests requirements, plans and procedures for compatibility and adequacy.
6. Monitoring of software tests for conformance with procedures, and verifying that test results are documented.
7. Implementation of a system for recording, reporting, and tracking software problems and for assuring the adequacy of corrective actions (49:49).

## How Much Quality Assurance Is Enough?

There are certain criteria that can be used in determining increased or decreased software quality assurance effort. One is to perform a risk analysis of the impact of the software on the overall program. Whenever risk is great,

an intensive quality assurance effort is merited (50:77). For example, the quality assurance needed for the National Aeronautics and Space Administration (NASA) space shuttle program is more critical, or higher risk, than for a remote radar tracking site.

Other risk factors which serve as criteria for increased emphasis on quality assurance include:

1. Complexity of software applications.
2. Amount of software (potential for error increases with size).
3. Stability of requirements.
4. Uniqueness of application. (Has this ever been done before?)
5. Lack of experienced personnel.
6. Rushed development schedules.
7. Mission criticality of the software.
8. Unavailability of realistic test environment (50:77-78).

Hardware vs. Software

Hardware quality assurance has been used successfully for many years. But even though the disciplines of hardware quality assurance may apply to software, a new set of conditions should be addressed by software quality assurance:

1. Software components do not degrade with time due to wear or fatigue.
2. Unlike hardware, software failures are seldom preceded by warnings.

3. The use of standard components is much more prevalent in hardware.
4. Hardware repairs restore the original condition; software repairs establish a new configuration state.
5. Hardware can usually be tested exhaustively; not so with software. There are many more distinct paths to check in software than in hardware.
6. Management has had a much better intuitive understanding of hardware than of software (25:11; 38:489-490).

## Life Cycle Models

Referring to a life cycle of software is the most common method of addressing software development. A review of the literature has produced an abundance of illustrations of the true or ideal life cycle. For the purpose of this study, the following six phases will be considered a software life cycle: Requirements Analysis, Preliminary Design, Detailed Design, Coding and Checkout, Testing and Integration, and Performance or Operation (6). Figure 3 shows the software life cycle and the key outputs of the phases.

The first phase to discuss is Requirements Analysis. During this phase both system and software requirements and their relationship are evaluated. Through analysis a determination is made as to what the software is to do in terms of inputs, processing, outputs, and accuracy. The following is a representative set of criteria (13:88) which may be applied during analysis:

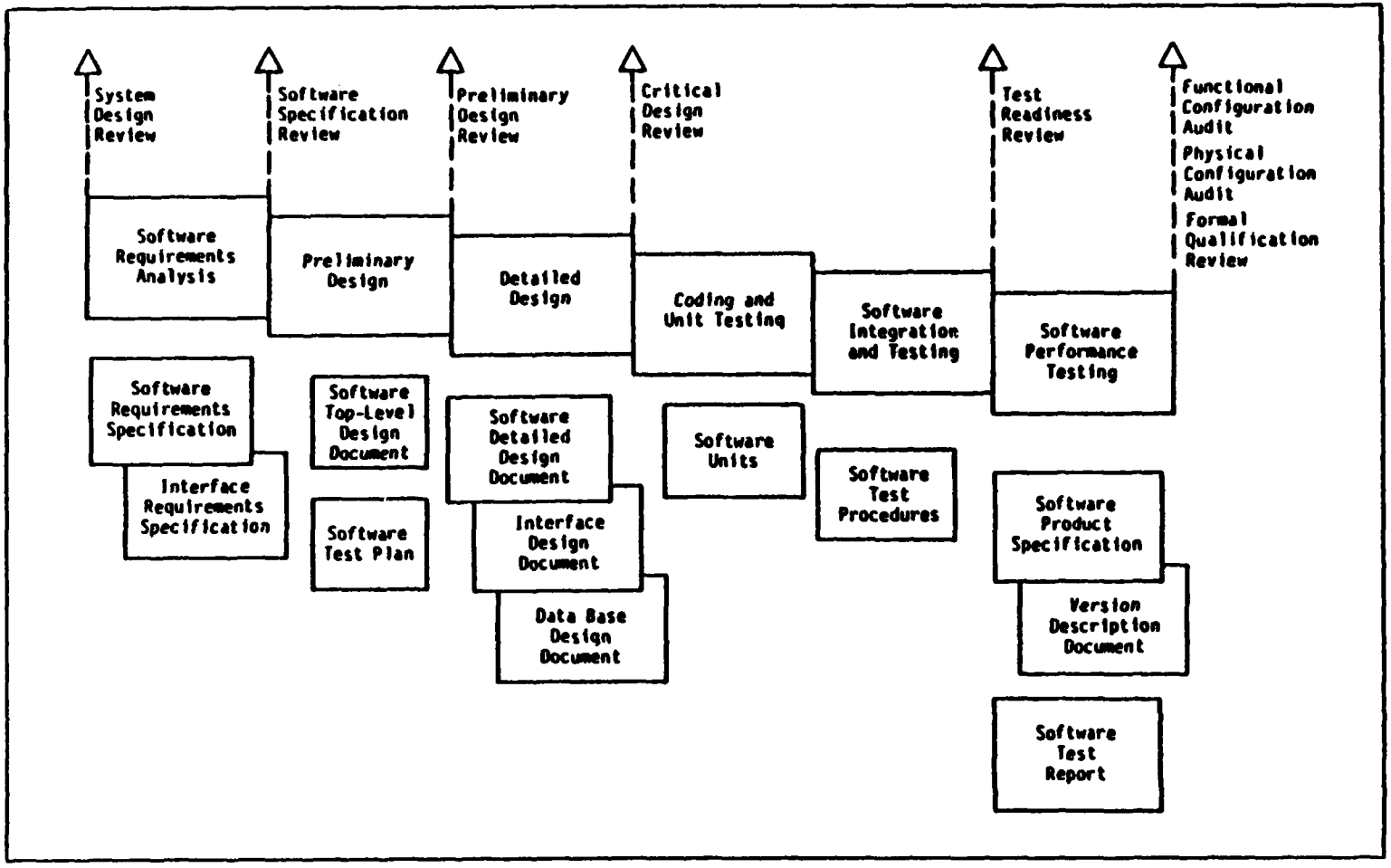



Figure 3. Software Life Cycle (6)

1. Realistic: Requirements must be achievable within the capabilities of the computer hardware.
2. Unambiguous: Requirements must be stated such that they are definitive and not open to subjective interpretations.
3. Consistent: Requirements must be consistent with one another, with interfacing subsystems, and with those at next higher and lower levels.
4. Necessary: Unnecessary or overly restrictive requirements will increase the cost and complexity of the software and will also impact the design, code, and testing schedules.
5. Complete: The requirements must completely specify the software product to be orovided in terms of accuracy, timing, throughput, interface control, and input/output.

The next phase is Preliminary Design. The object of this phase is to translate the requirements into a software design. The design will include mathematical models, the allocation of the requirements to components of the software, the relationship between these components, and the external interfaces (6).

Following Preliminary Design is the third phase, Detailed Design. The purpose of Detailed Design is to extend the level of detail by identifying individual software units. These units must be defined in sufficient detail for coding and unit testing. Also, they include functional descriptions, logical flows, algorithms, and constraints (6).

The fourth phase, Coding and Checkout, includes translating the detailed design into a computer programming language. Usually it is a high-order language but it may also be assembly language. Once compilation and assembly errors are corrected, each individual software unit is executed to remove obvious defects, and software tools are used to remove the not so obvious defects. This procedure is considered the Checkout (13:09-91).

Once coding is complete, the Testing and Integration Phase begins. Here the developed software is tested to show consistency with system and software requirements. The integration portion of this phase typically involves two forms of integration. First, individual software units are synthesized into subsystems and then systems. Second, the entire software system is integrated with the computer hardware (8:23).

The last phase of the cycle is the Performance Phase or Operational Phase. This phase is conducted to determine if the software indeed satisfies the specified requirements. The key element of this phase is the environment. Any testing and evaluation should be conducted in an environment as operationally realistic as possible (6). Figure 4 shows the relationship between a software life cycle and a hardware life cycle within a major system development cycle.

Throughout the software life cycle there is on-going assessment being performed commonly known as "verification and validation."

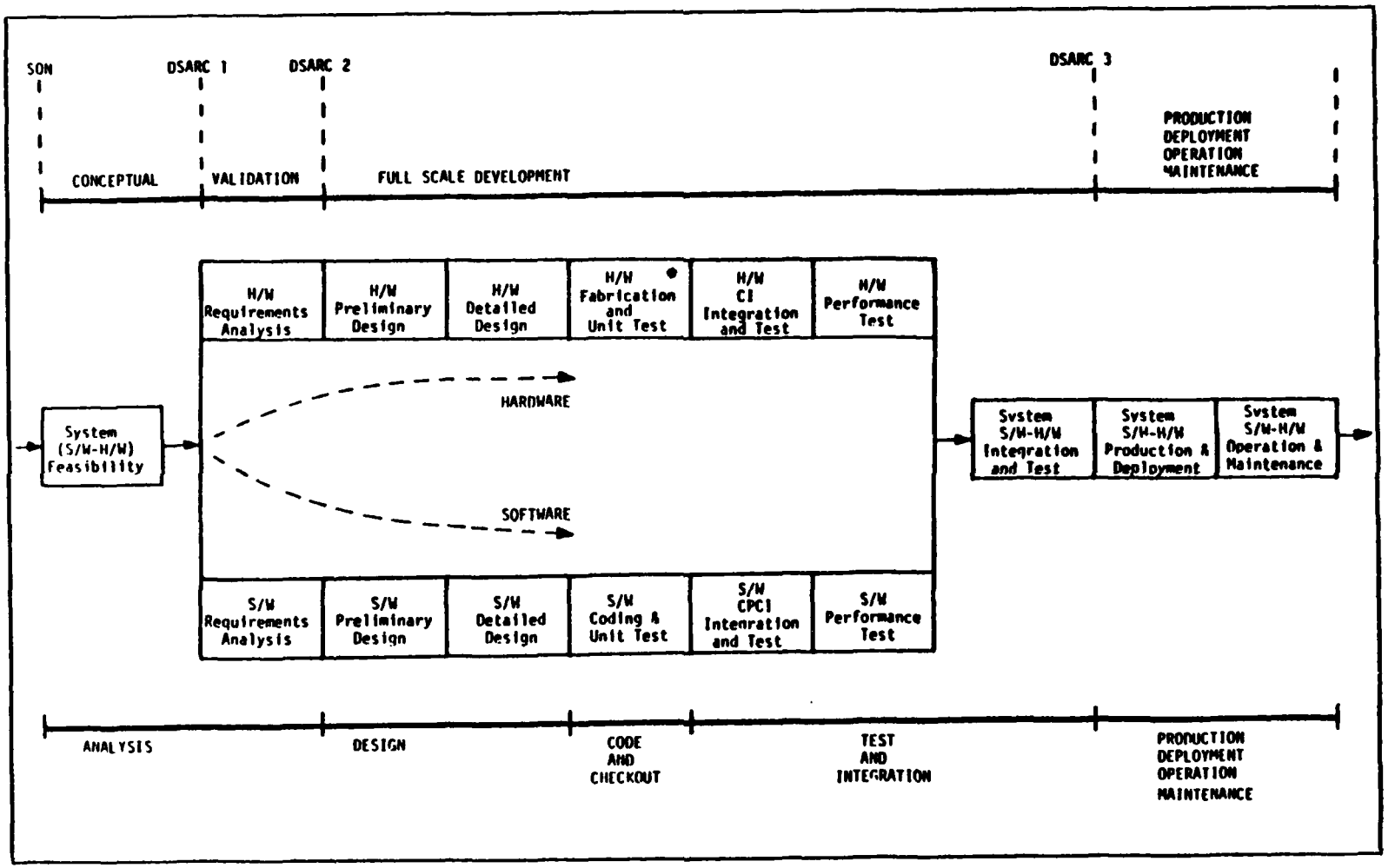



Figure 4. Idealized Software-Hardware System Life Cycle (6)

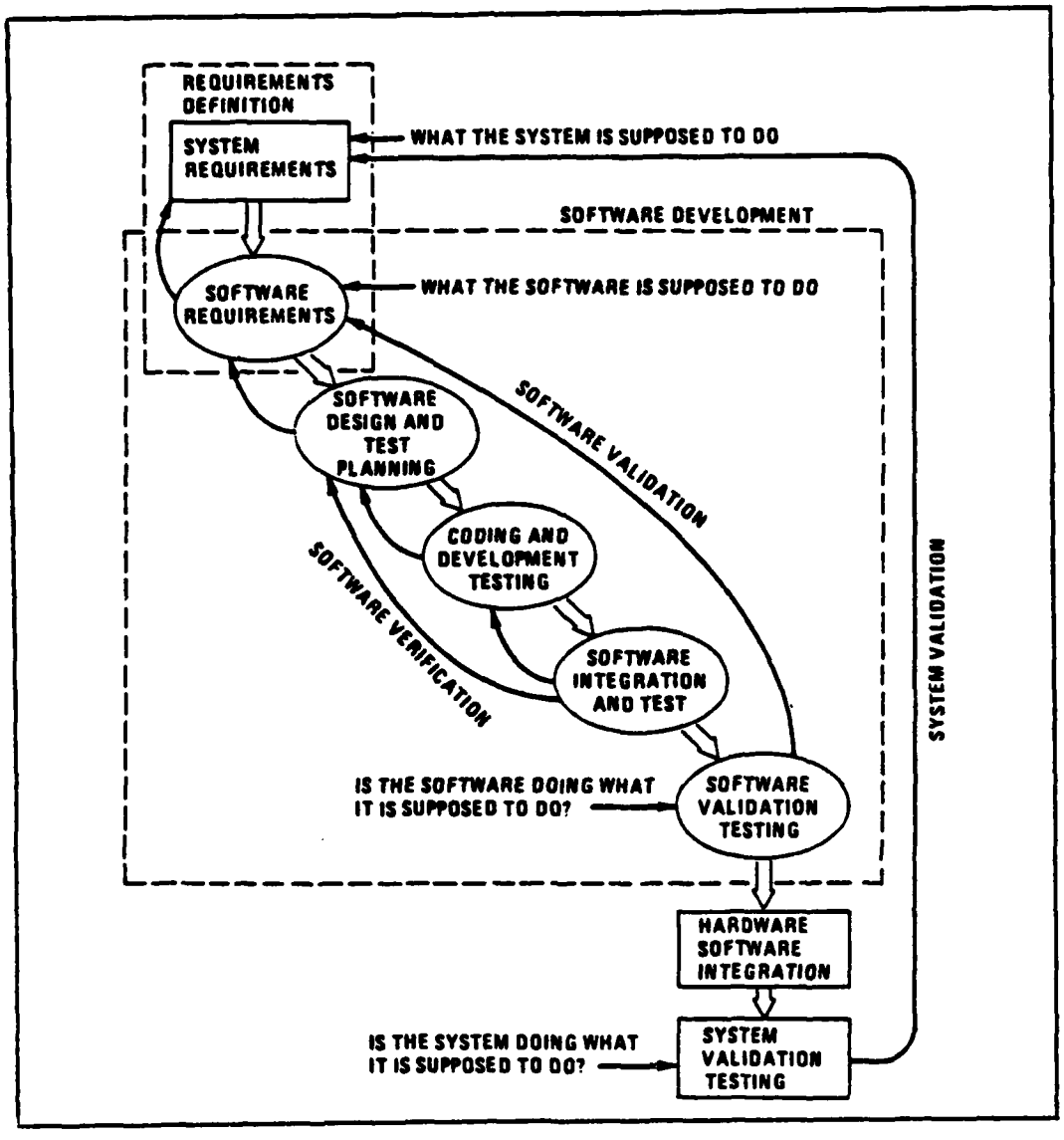



Figure 5. Software Verification and Validation (13:86)

> Verification and validation is the systematic process of analyzing, evaluating, and testing system and software documentation and code to ensure the highest possible quality, reliability, and satisfaction of system needs and objectives (37:238).

In the past, this activity has proven to be a very effective means for achieving high-quality software (45:659). As illustrated in Figure 5, verification is the evaluation process

designed to ensure consistency and completeness of the product at any phase within the software life cycle. Consistency is concerned with measuring the degree to which a phase is in agreement with the previous phase. Completeness is a measure of the readiness to continue into the next phase. Also illustrated, validation is directed at test and evaluation to measure how well the product performs against established requirements (13:85).

Including software quality assurance within all phases of the software life cycle is essential to the development of quality software (32:32). During the Requirements Analysis Phase, quality assurance should assist in the review of requirements to determine acceptability. During the Design Phase, quality assurance should work with software development personnel to recommend and maintain standards, procedures, and plans that affect the remainder of the life cycle. During the Code and Checkout Phase, numerous monitoring functions, such as software code review, are conducted to verify compliance with standards. During Testing and Integration, quality assurance should review test plans and procedures. During the Operational Phase, quality assurance performs a final review to determine whether all plans and procedures were in accordance with the requirements of the user (49).

The following are benefits of including quality assurance in a software life cycle:

1. Life cycle models with software quality assurance activities integrated can serve as planning guides so that important areas are not omitted.
2. Fully developed life cycle models, with integrated quality assurance, provide a method of keeping track of which activities are to be performed, and reports that are to be written.
3. Software quality assurance personnel working with a life cycle model will review and evaluate the approach, the methods, the status, and the achievements during each phase of the software development. This allows for early detection and timely correction of problems.
4. A life cycle model provides a significantly practical tool for training software quality assurance personnel (49:50-52).

Quality Factors and Criteria

The concept of software quality originates from a set of attributes. Explicit attention to these attributes can lead to significant savings in software cost (10:218). Therefore, an understanding of the attributes of software quality can lead to a better understanding of software cost. For the purpose of conformity, the terms attributes, characteristics, and factors have the same meaning when referring to software quality (55; 10; 40).

Another reason for understanding software quality factors is because they provide a means to define quality requirements.

TABLE 1

Definition of Software Quality Factors (40:129)

| | |
|---|---|
| *Correctness* | Extent to which a program satisfies its specifications and fulfills the user's mission objectives. |
| *Reliability* | Extent to which a program can be expected to perform its intended function with required precision. |
| *Efficiency* | The amount of computing resources and code required by a program to perform a function. |
| *Integrity* | Extent to which access to software or data by unauthorized persons can be controlled. |
| *Usability* | Effort required to learn, operate, prepare input, and interpret output of a program. |
| *Maintainability* | Effort required to locate and fix an error in an operational program. |
| *Testability* | Effort required to test a program to insure it performs its intended function. |
| *Flexibility* | Effort required to modify an operational program. |
| *Portability* | Effort required to transfer a program from one hardware configuration and/or software system environment to another. |
| *Reusability* | Extent to which a program can be used in other applications—related to the packaging and scope of the functions that programs perform. |
| *Interoperability* | Effort required to couple one system with another. |

This is important because experience has shown that poor definition of requirements is a source of software design, test, and operational problems (55:233). Refer to Table 1 for software quality factor definitions.

The quality factors in Table 1 can be further broken down into criteria. These criteria further define the quality factor and help describe the relationship between factors. The criteria are independent attributes of the software by which the quality can be judged, defined, and measured (40:130). The relationship of criteria to software

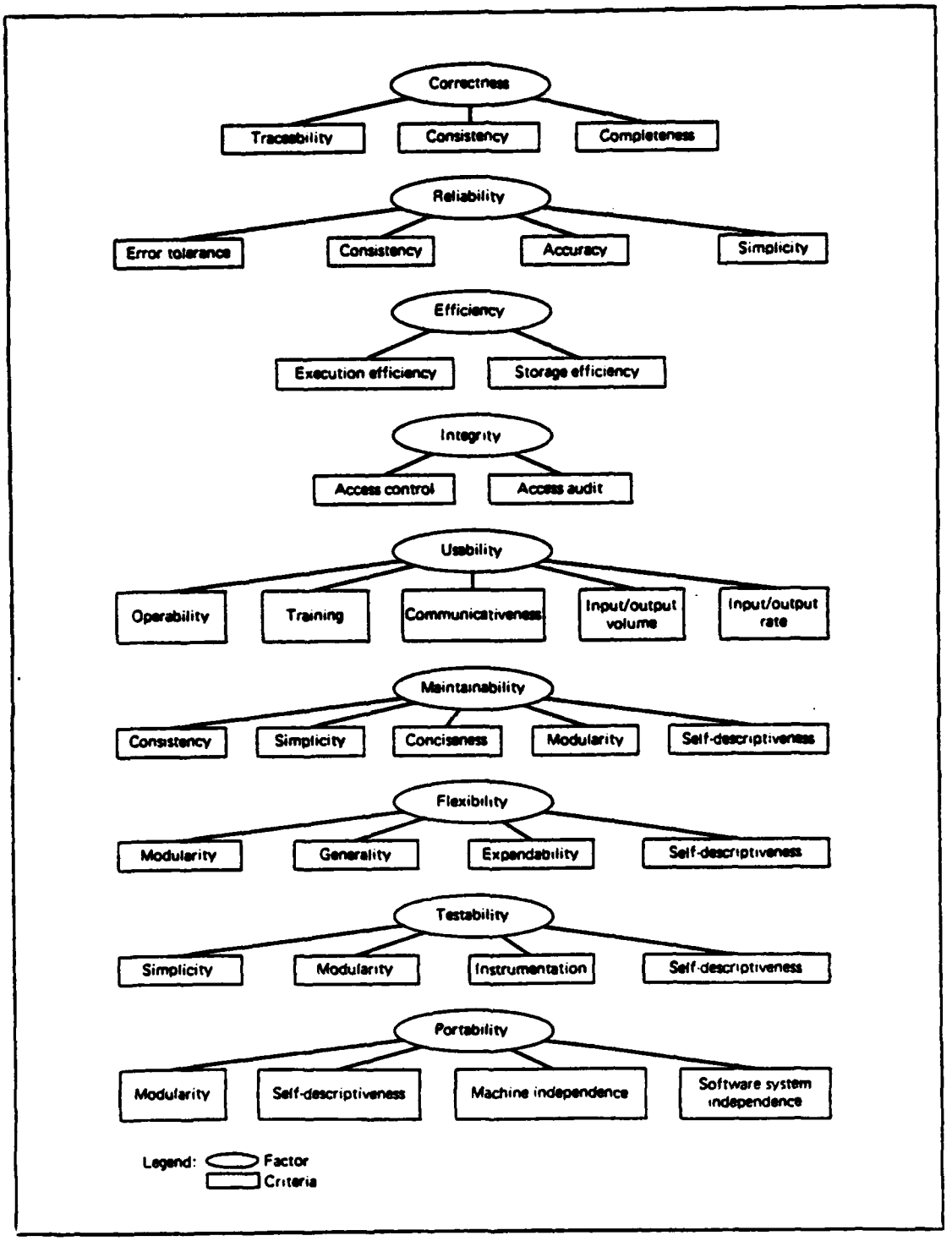



Figure 6. Relationship of Criteria to Software Quality Factors (40:131-132)

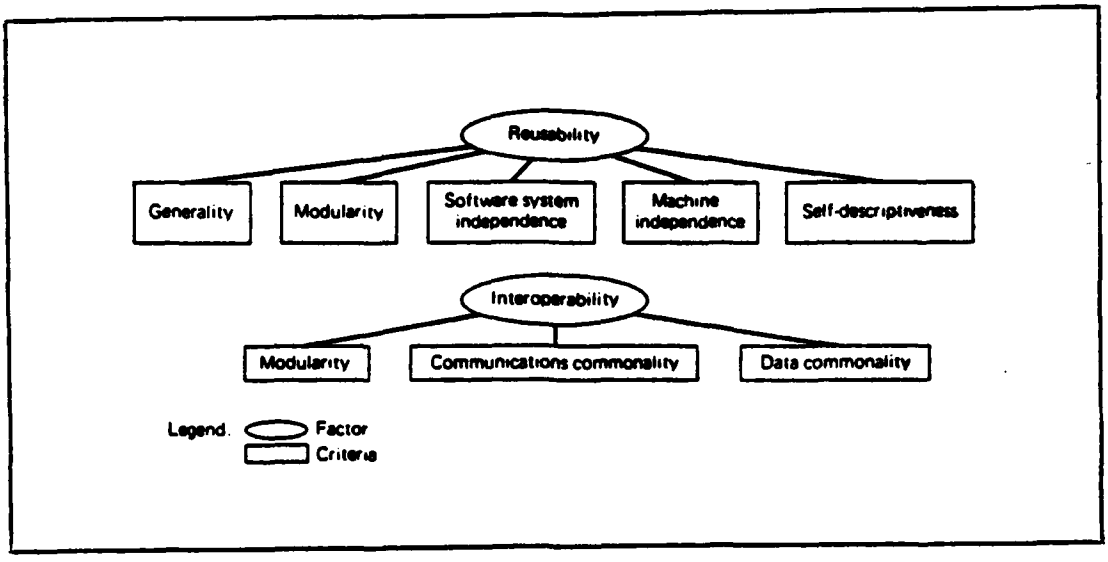



Figure 6. Continued

quality factors are shown in Figure 6. The definitions of these criteria are provided in Table 2.

An important consideration in identifying software quality factors is some of them are in conflict with others (51:62). Figure 7 can be used as a guide in determining these conflicts. The following are examples of conflicts:

1. The use of assembly language and exploitation of special hardware features can enhance Efficiency at the expense of Portability.
2. The use of detailed error messages enhance Testability at the expense of Efficiency (51:62).

Quality Assurance Standards

In the past, the establishment of conformance standards has been neglected for software development. Finally, this

TABLE 2

Criteria Definitions for Software Quality Factors
(40:133-134)

| *Criterion* | *Definition* | *Related Factors* |
|---|---|---|
| *Traceability* | Those attributes of the software that provide a thread from the requirements to the implementation with respect to the specific development and operational environment. | Correctness |
| *Completeness* | Those attributes of the software that provide full implementation of the functions required. | Correctness |
| *Consistency* | Those attributes of the software that provide uniform design and implementation techniques and notation. | Correctness<br>Reliability<br>Maintainability |
| *Accuracy* | Those attributes of the software that provide the required precision in calculations and outputs. | Reliability |
| *Error Tolerance* | Those attributes of the software that provide continuity of operation under non-nominal conditions. | Reliability |
| *Simplicity* | Those attributes of the software that provide implementation of functions in the most understandable manner. (Usually avoidance of practices which increase complexity.) | Reliability<br>Maintainability<br>Testability |
| *Modularity* | Those attributes of the software that provide a structure of highly independent modules. | Maintainability<br>Flexibility<br>Testability<br>Portability<br>Reusability<br>Interoperability |
| *Generality* | Those attributes of the software that provide breadth to the functions performed. | Flexibility<br>Reusability |
| *Expandability* | Those attributes of the software that provide for expansion of data storage requirements or computational functions. | Flexibility |
| *Instrumentation* | Those attributes of the software that provide for the measurements of usage or identification of errors. | Testability |
| *Self-Descriptiveness* | Those attributes of the software that provide explanation of the implementation of a function. | Flexibility<br>Maintainability<br>Testability<br>Portability<br>Reusability |

TABLE 2

Continued

| *Criterion* | *Definition* | *Related Factors* |
|---|---|---|
| *Execution Efficiency* | Those attributes of the software that provide for minimum processing time. | Efficiency |
| *Storage Efficiency* | Those attributes of the software that provide for minimum storage requirements during operation. | Efficiency |
| *Access Control* | Those attributes of the software that provide for control of the access of software and data. | Integrity |
| *Access Audit* | Those attributes of the software that provide for an audit of the access of software and data. | Integrity |
| *Operability* | Those attributes of the software that determine operation and procedures concerned with the operation of the software. | Usability |
| *Training* | Those attributes of the software that provide transition from current operation or initial familiarization. | Usability |
| *Communicativeness* | Those attributes of the software that provide useful inputs and outputs which can be assimilated. | Usability |
| *Software System Independence* | Those attributes of the software that determine its dependency on the software environment (operating systems, utilities, input/output routines, etc.). | Portability<br>Reusability |
| *Machine Independence* | Those attributes of the software that determine its dependency on the hardware system. | Portability<br>Reusability |
| *Communications Commonality* | Those attributes of the software that provide the use of standard protocols and interface routines. | Interoperability |
| *Data Commonality* | Those attributes of the software that provide the use of standard data representations. | Interoperability |
| *Conciseness* | Those attributes of the software that provide for implementation of a function with a minimum amount of code. | Maintainability |

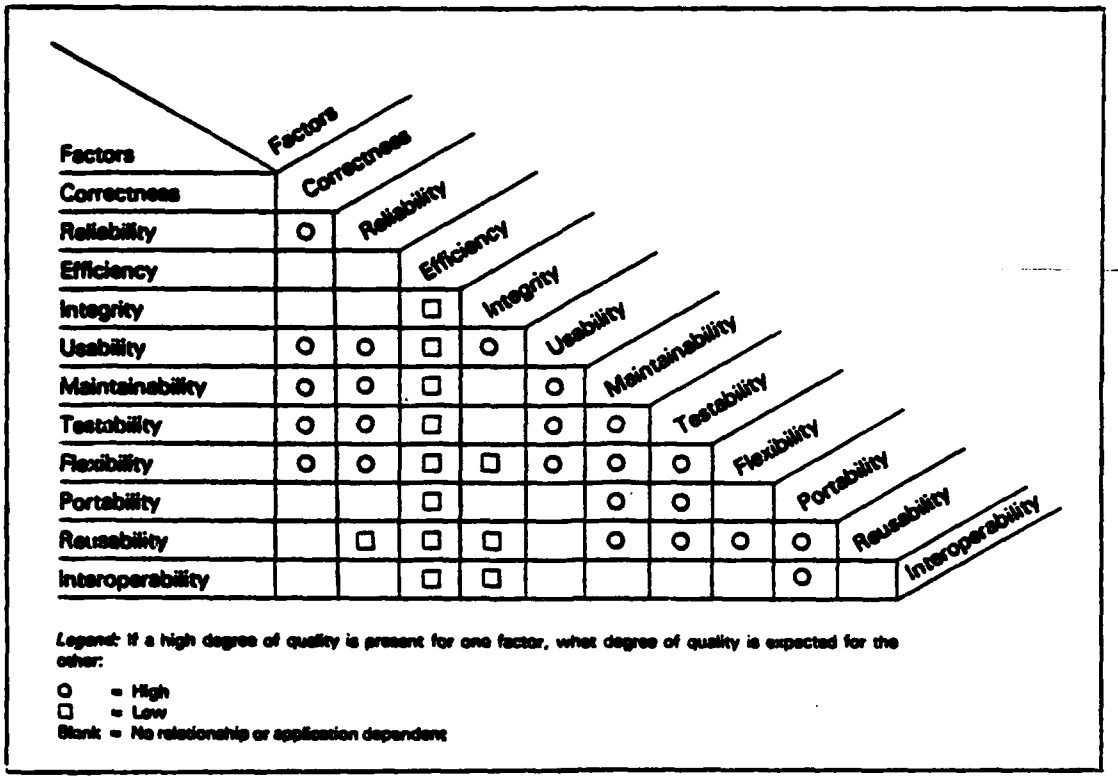



Figure 7. Relationships Between Software Quality Factors (53:147)

shortcoming was recognized and attempts are being made to provide adequate standards (39:19).

IEEE Policy. The Institute of Electrical and Electronics Engineers (IEEE) established a subcommittee to develop a software quality industry standard, IEEE Standard 730. The purpose of this standard was to "provide uniform minimum acceptable requirements for the preparation and content of Software Quality Assurance Plans" (12:45). The concern here is that the basic plan of developing the software should be well defined. This standard does not dictate

what approach is to be used; however, it does identify common elements required for quality assurance.

Directives, Military Standards/Specifications, and Regulations. The following list outlines current policy that applies to software quality assurance. This is not an exhaustive list, but instead is a list of "most commonly used" policies:

| | |
|---|---|
| DODD 5000.29 | Management of Computer Resources in Major Defense Systems |
| MIL-S-52779A | Software Quality Assurance Program Requirements |
| MIL-Q-9858A | Quality Program Requirements |
| MIL-STD-480 | Configuration Control-Engineering Changes, Deviations and Waivers |
| MIL-STD-483 | Configuration Practices for Systems, Equipment, Munitions, and Computer Programs |
| MIL-STD-490 | Specifications Practices |
| MIL-STD-1521A | Technical Reviews and Audits for Systems, Equipment, and Computer Programs |
| AFR 800-14 Vol. I | Management of Computer Resources in Systems |
| AFR 800-14 Vol. II | Acquisition and Support Procedures for Computer Resources in Systems |

The availability of information does not ensure that high quality software will be developed. This information must be used and faithfully followed. Appendix A gives a more comprehensive list of government documents which may be used in the quality assurance effort.

Test Oriented Quality Assurance. This type of quality assurance is illustrated by provisions in MIL-STD-483 and

MIL-STD-490. Basically, these provisions equate quality assurance to a test program. They specify test plans and procedures, categories and types of tests, and methods of formal verification of a design requirement as part of a test activity (19; 21).

The major pitfall of test oriented quality assurance is you don't test quality into a software product, you design and build it in. If a serious quality defect is discovered during a formal test phase, it may be too late or too expensive to properly correct it (51:2; 55:230).

Development Oriented Quality Assurance. Quality assurance of this kind is best described as a narrow interpretation of MIL-S-52779A. In other words, it means to assure that the software delivered under contract complies with the requirements of the contract.

The pitfall of this approach is that if the contract specifies poor quality software, the quality assurance program will also assure that you get poor quality software. You get what you ask for and pay for.

To offset this weakness, MIL-S-52779A must be supplemented by a great deal of early planning and quality specifications (51:2-3; 55:230).

Life Cycle Oriented Quality Assurance. This style of quality assurance is exemplified by an expanded interpretation of MIL-S-52779A as supported by the life cycle principles from DODD 5000.29 and AFR 800-14. These documents

properly focus effort on the early requirements phases as the appropriate place for planning and specifying attributes of software quality. Life cycle oriented quality assurance provides the opportunity to reduce costs by assuring quality software during software development (51:3; 55:231).

Joint Service Policy. The Department of Defense started a number of initiatives in the mid-1970s aimed at providing better management of software. The services then developed their own policies for acquiring and supporting computer resources (36:191).

In January 1979, a subgroup of the Joint Logistics Commanders (JLC) planned and conducted a software workshop to determine if there was a basis for coordination and adoption of joint service policy and standards (36:192). The subgroup concluded that the services should develop common policy, development standards, and documentation standards instead of continuing to approach software development in a service unique manner (39:19).

To further satisfy the purpose of the workshop, the subgroup defined a program to develop a military standard for software development, MIL-STD-SDS, and a set of changes to MIL-STD-483, 490, and 1521A (39:20). In addition, software documentation standards were developed that identified the types of documents needed to support mission critical computer resource software (36:200). At this time, there is not an approved or coordinated position of the Joint Logistic Commanders.

The effort to adopt common joint service policy and standards is well under way. The potential cost savings are significant. "No longer will industry be required to maintain multiple-management control systems. By providing a more uniform environment for software development, joint service programs should run more smoothly" (36:200).

## Software Quality Assurance Program

### Quality Assurance Planning.

Planning involves the

> . . . developing and formulating a course of action, and the output results constitute a plan. It is the systematic identification of program tasks, task schedules, and the resources required for task accomplishment. Planning is necessary to achieve some degree of order in completing activities scheduled in the immediate future and for long-range activities that are forecasted far out in time (7:47).

According to the definition above, the output of quality assurance planning is the formation of a plan. On a broad scope, a quality assurance plan must indicate the particular activities that will achieve the required level of quality. This quality plan should include at least the following topics:

1. Organization: The organizational approach includes the definition of roles and responsibilities of each group in the organization. The independence and reporting lines of the quality assurance group must be clearly established.
2. Requirement Traceability: This defines the methodology to assure the requirements in top level specifications are satisfied in the lower level specifications.

3. Documentation: The documentation must be defined to assure formal, controlled communication, standards for the document preparation, and the measurement of compliance with standards.
4. Techniques and Tools: The application and verification of the quality related techniques and tools must be defined.
5. Formal Reviews: A definition of the reviews must be made to assure readiness and how the reviews are to be accomplished.
6. Test Program: The plan must specify the measures for technical review of the test procedures and compliance with prescribed standards.
7. Configuration Management: The quality assurance considerations include a software library with control procedures to assure identification of the products and prevention of unauthorized modifications, and definition of procedures for the generation, disposition, tracking and closeout of design and test discrepancies (48:195).

Two documents give assistance in developing software quality assurance plans, MIL-S-52779A and IEEE Standard 730. The first document, MIL-S-52779A, applies to the acquisition of software either alone or as part of a complete system. It requires the establishment and implementation of a software quality assurance program. Figure 8 illustrates a software quality assurance plan as proposed by MIL-S-52779A

1. Work Tasking and Authorization Procedures

    a. Procedures and Schedules
    b. Work Descriptions
    c. Status Reports
    d. Resources Estimates

2. Configuration Management

    a. Baseline Establishment
    b. Change Accountability
    c. Audits

3. Testing

    a. Analysis to Determine Testability
    b. Test Plan/Procedure Review
    c. Monitor and Certification of Test Results
    d. Tests vs. Requirements Traceability

4. Corrective Action

    a. Problem Reporting and Measuring
    b. Trend Analysis
    c. Corrective Action Assessment

5. Library Controls

    a. Code Control and Related Documentation
    b. Media Identification and Protection
    c. Change Control

6. Computer Program Design Review

    a. Contract Compliance
    b. Evaluation of Design Logic

7. Software Documentation

8. Reviews and Audits

9. Tools, Technologies, Methodologies

10. Subcontractor Controls

Figure 8. MIL-S-52779A Software Quality Assurance Plan (17)

1. Purpose

2. Reference Documents

3. Management

   a. Organization
   b. Tasks
   c. Responsibilities

4. Documentation

   a. Purpose
   b. Minimum Required Documentation: Software Requirements Specification, Software Design Description, Software Verification Plan
   c. Other: Computer Program Development Plan, Configuration Management Plan, Standards and Procedures Manual

5. Standards, Practices and Conventions

   a. Purpose
   b. Content: Document Standards, Logic Structure Standards, Coding Standards, Commentary Standards

6. Reviews and Audits

   a. Purpose
   b. Minimum Requirements: Software Requirements Review, Preliminary Design Review, Critical Design Review, Functional Audit, Physical Audit, In-Process Audits

7. Configuration Management

8. Problem Reporting and Corrective Action

9. Tools, Techniques and Methodologies

10. Code Control

11. Media Control

12. Supplier Control

Figure 9. IEEE Standard 730 Quality Assurance Plan (12)

The second document to assist the planning effort is IEEE Standard 730. This standard applies to the development and maintenance of critical software (i.e., where failure could impact safety or cause large financial or social losses). Figure 9 illustrates a plan under IEEE Standard 730. An early involvement in the planning process is the key to a successful software quality assurance program (46:497).

Staffing and Organization. The success of any quality assurance program begins with the personnel assigned to the project. The most effective means of obtaining knowledgeable quality assurance personnel is through the transfer of well-respected and competent people from the development group. This has many advantages, such as familiarity with the type of software developed and procured, insight into known weaknesses, rapport with in-house designers, and the knowledge of how to operate within the organization (25:269).

It is not enough for quality assurance personnel to merely possess the characteristics, for example, of good systems analyst. They must have the respect of peers and all levels of management with whom they must interface. Also, they must have good communication skills and be able to use logical persuasion. The success of the organization will depend on the individual's ability to sell his ideas for improvement (35:95).

An organization can set up a quality assurance department in many ways. Figure 10 depicts a simplified view of an organization with quality assurance placed as a staff

function reporting to the General Manager. This structure ensures that quality assurance will receive the attention of the General Manager and ensures independence from other departments. Figure 11 depicts a quality assurance department embedded in the overall organizational structure of the project organization. With this type of structure, coordination between projects and even with the General Manager becomes difficult or time consuming. Figure 12 depicts a functional organization with quality assurance placed on the same level as other departments. This structure, with quality assurance as an independent function reporting directly to the General Manager, provides good independent oversight. As another example, Figure 13 depicts a matrix organization. Under this type of structure, there is a top-level quality department and lower-level quality assurance departments for each project. A matrix organization allows independent checks for each project falling under the General Manager's control.

The need for independence cannot be stressed too highly. Many times software is developed under the most demanding of schedules. Requirements may be ill-defined or late, completion of the systems analysis may slip, or hardware availability may be delayed. Only one thing never changes, the "end date." Under these circumstances there is pressure to skimp on test planning, various documentation of the design, library control procedures, test documentation, reviews and audits. Only the devotion to quality of an independent

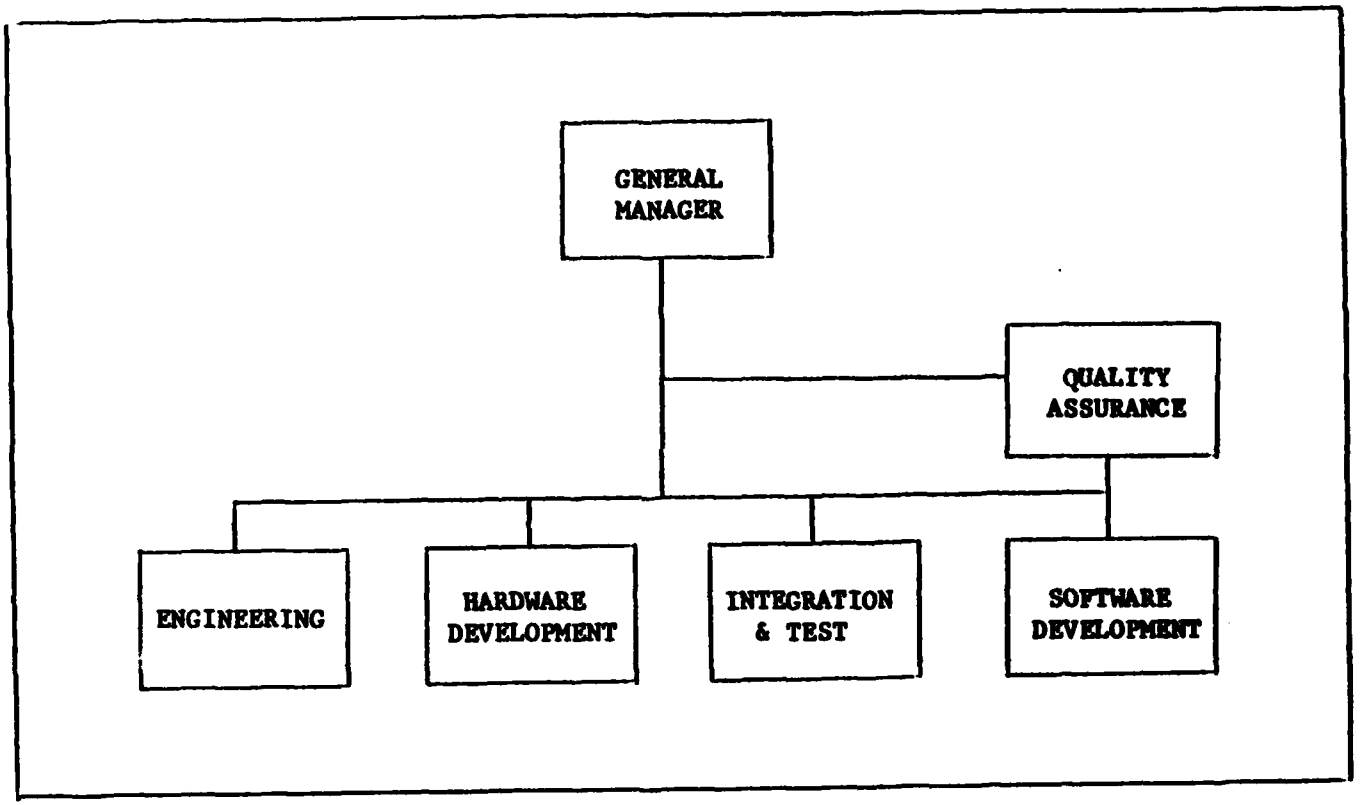



Figure 10. Staff Organizational Structure

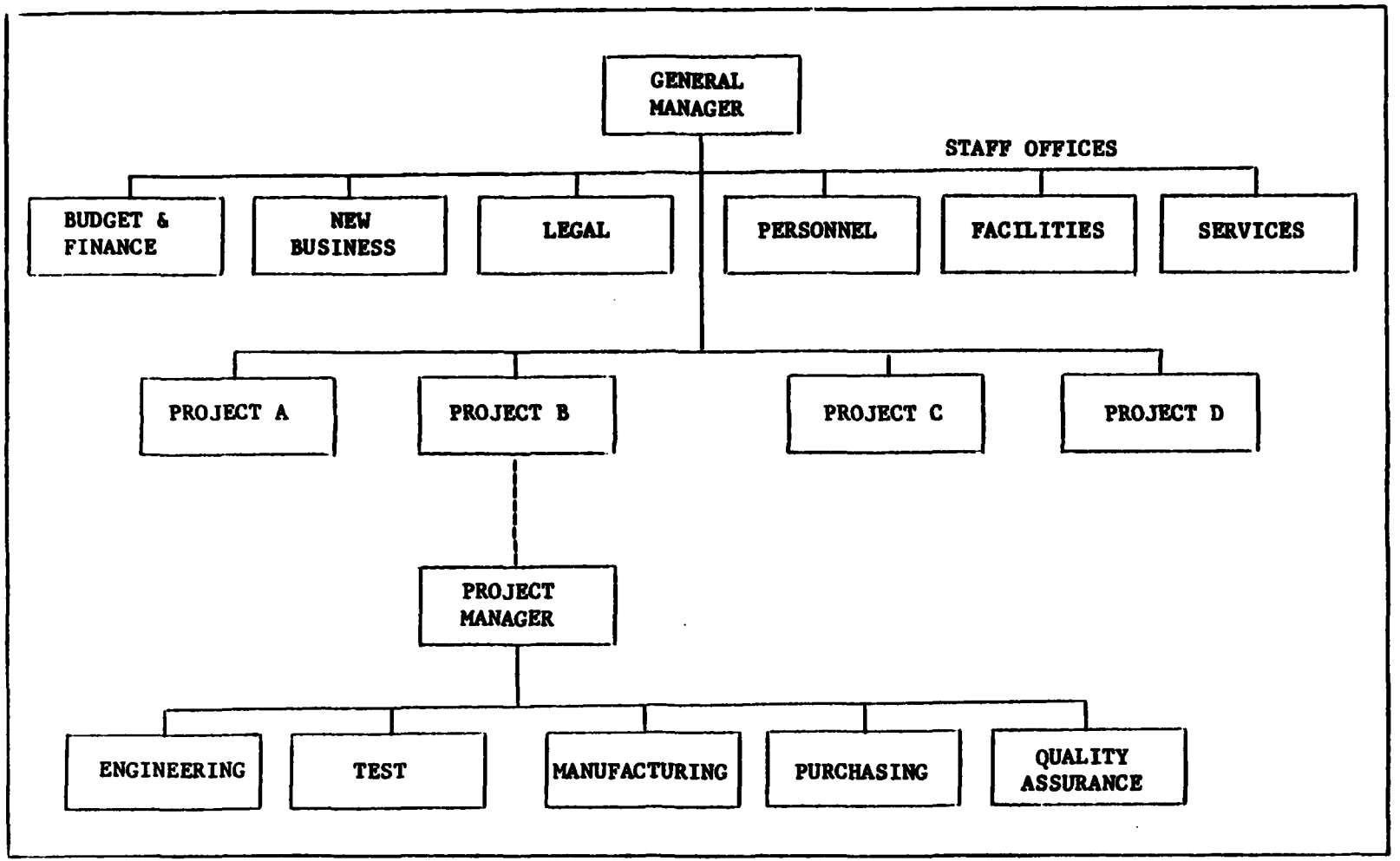



Figure 11. Project Organizational Structure (6)

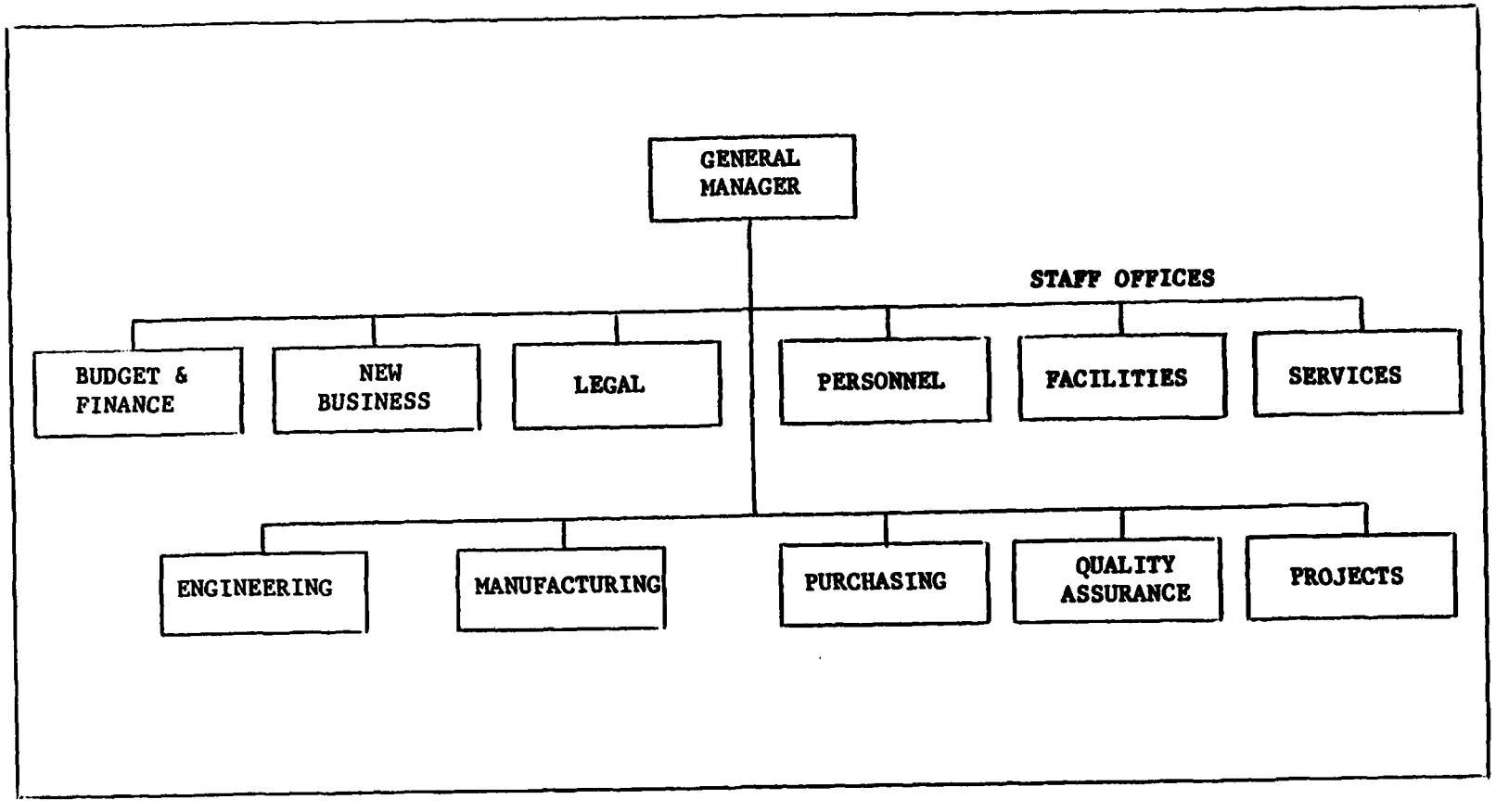



Figure 12. Functional Organizational Structure (6)

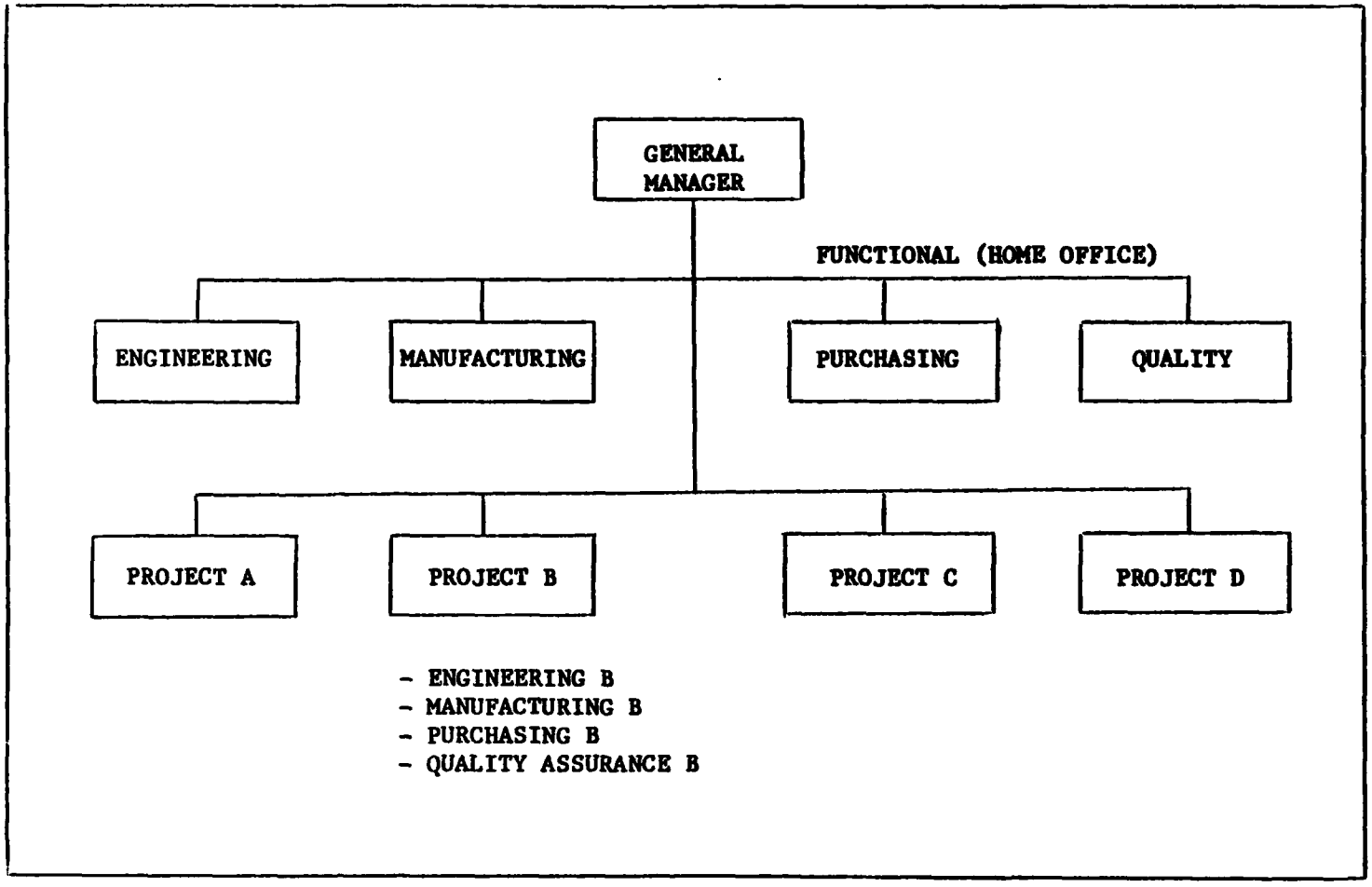



Figure 13. Matrix Organizational Structure (6)

functional activity can stop those who might compromise quality in an attempt to recover schedule slippages or cost overruns. Independence is the key (25:271-272).

Reviews and Audits. Controlling involves "the monitoring of program activities to ensure that the end objectives are being met" (7:48). Thus, controlling means to use checks and balances on a periodic basis so that problem areas can be detected as early as possible in the software development process. Basically, this is accomplished through a combination of reviews and audits.

The purpose of a review is to examine the system requirements and design to assure appropriate technical progress according to plan. The purpose of an audit is to determine, through investigation, compliance to and adequacy of established specifications and standards (6).

Naturally, the number of review points depend on a number of variables such as size of the system and makeup of the quality assurance organization. But the general consensus is that reviews must be predefined, occur at key points in the development process, be understandable and complete, and are conducted in accordance with prescribed standards.

The IEEE Standard 730 proposes a certain minimum number of reviews which should be conducted during the software life cycle (12:48). These include:

1. A software requirements review to ensure the adequacy of the requirements stated in the software requirements specifications.

2. A preliminary design review to evaluate the technical adequacy of the preliminary design of the software.
3. A critical design review to determine the acceptability of detailed software design.

The recommended audits consist of:

1. A functional audit, held prior to software delivery, to verify compliance with all requirements specifications.
2. A physical audit to verify that the software and documentation are internally consistent and ready for delivery.
3. In-process audits to verify consistency of the design, such as peer audits.

Several Air Force regulations, specifications, and standards provide guidance on conducting reviews and audits. In particular, MIL-STD-1521A provides guidance for the following formal reviews and audits (16):

1. System Requirement Review (SRR): Conducted to determine initial direction and progress in defining system requirements.
2. System Design Review (SDR): Concerned with evaluation of the total system requirements.
3. Preliminary Design Review (PDR): A review of the top level software design in response to the software specifications.
4. Critical Design Review (CDR): Concerned with the crucial review of the detailed design of the software prior to the start of software coding.

5. Functional Configuration Audit (FCA): Ensures that the delivered computer code actually does what is asked for in the specifications.
6. Physical Configuration Audit (PCA): Ensures that the support documentation accurately and clearly relects the software.
7. Formal Qualification Review (FQR): Verifies that the actual performance of the software as determined through test complies with its specifications.

Using these reviews and audits, the developing organization's technical progress can be monitored. They also reveal the technical progress of the phases in the software life cycle. Referring back to Figure 3 gives a basic look at how the reviews and audits fit into a software life cycle.

Software Testing. The test design, which forms the testing philosophy, is usually based on two methods, bottom up and top down (6). Bottom up testing begins by testing each software routine or module as it becomes available from the development group. Routines are then combined and tested until the entire software product has been tested. Top down testing begins by integrating and testing the highest level routines. After these routines are integrated, the process is repeated for routines at the next lower level until all the software is successfully integrated. The advantages and disadvantages of each approach are summarized in Table 3.

Unit testing, performed by the software development group on each routine, is the lowest of software testing.

TABLE 3

Advantages and Disadvantages of Bottom Up and Top Down (6)

| Bottom Up | |
|---|---|
| **Advantages** | **Disadvantages** |
| High risk routines tested early | Testing difficult because interface problems and system requirements not addressed early |
| Utility routines are developed early and tend to be common to the program | Hard to maintain visibility of entire software system |
| Easier to control testing conditions | Difficult to change because interfaces may be "kludged" |
| Development of top level structure can be delayed, allowing selection of a specific machine to be made later | Data base structure not addressed early can result in "kludged" data base |
| Dummy routines simple to develop | System cannot be executed until late in testing |
| **Top Down** | |
| **Advantages** | **Disadvantages** |
| System executes early in development and testing | Low level high risk modules not developed early |
| Data structures and interface problems addressed and resolved early | Multiple utility routines may exist or redesign may be needed to use common utility routines |
| Testing can be done in parallel at more than one level | Developing top level structure early may force project to specific machine(s) too early |
| Easier to maintain visibility of the entire software system | Dummy routines must be developed |
| | Testing conditions may be hard to control |
| | Coding of top level routines may begin before the design is completed |
| | Error conditions may be more difficult (costly) to exhaustively exercise |

It is intended to make each executable section of code, routine or module, as error-free as possible and is oriented to finding errors which commonly occur in the development of software (6).

After routines have successfully completed unit testing, they are combined and executed in integration tests. In a typical project, the responsibility for software integration testing will be divided between the development and testing groups. The lowest level of testing may be conducted by the programming teams that developed the software. Tests not conducted by the programming teams are executed by individual test teams (6).

Independence of the test team becomes important here, and can be accomplished by using an independent verification and validation (IV&V) process. During the IV&V activity, deficiencies will be discovered. The software development group must not receive direction form the IV&V team to correct the deficiencies. Rather, the deficiencies should be given to the quality assurance management for action. This is done to preserve the independence of the IV&V team (45:660).

Tests performed by the test teams require more formal documentation of the testing effort. Four types of documents are usually required: test plans, test procedures, test execution reports, and test results analysis reports. These documents encompass the planning, execution, and analysis of the test process (13:92).

Software quality assurance should become involved in testing in a number of areas. Before the testing begins, quality assurance should ensure that all software, hardware, and the testing environment are under control. It should witness loading and running of the software and ensure the test results are retained and discrepancies noted. Finally, quality assurance should participate in the post-test analysis and certify the test report on satisfactory completion (48:198).

MIL-S-52779A, the specification for software quality assurance, contains a comprehensive list of software testing procedures. These procedures consist of:

1. Analysis of software requirements to determine testability.
2. Review of test requirements and criteria for adequacy, feasibility, and traceability and satisfaction of requirements.
3. Review of test plans, procedures, and specifications for compliance with contractor and contractual requirements and to insure that all authorized and only authorized changes are implemented.
4. Verification that tests are conducted in accordance with approved test plans and procedures.
5. Certification that test results are the actual findings of the tests.
6. Review and certification of test reports.
7. Ensuring that test related media and documentation are maintained to allow repeatability of tests (17:3-4).

Many software producing organizations do not understand software testing techniques, or the importance of continuous

testing. But by increasing the awareness of software testing through training programs and encouraging test development throughout the software life cycle, these organizations can minimize problems with the software and maximize software quality (15:14).

Configuration Management. Configuration management consists of identifying the configuration of the software at discrete points in time. The purpose is to systematically monitor changes to this configuration and maintain the integrity and traceability of this configuration throughout the software life cycle (47:31). Quality assurance, through configuration management, should enforce the following:

1. Configuration Identification: The functional and physical characteristics, or configuration, of the software is identified by and documented in a series of specifications.
2. Configuration Control: In the configuration control process, changes to the established software specifications are classified, evaluated, approved or disapproved, released, implemented, and verified. The purpose is to assure that the software configuration used in critical phases of testing, acceptance, and delivery is known and compatible with the specifications.
3. Configuration Status Accounting: Status accounting is the recording and reporting of data concerning the software's configuration identification, proposed changes to

its configuration identification, and the implementation status of approved changes (52:1.4).

Library Control. A key element in the quality assurance program is the software library which provides visibility and control of software and software documentation. Documentation and software storage, retrieval and change processing are essential activities in a software library (51:28).

A subcategory of library control is concerned with organization and protection of software media (e.g., card decks, magnetic tape and disk). Loss of information due to defective media can be disastrous and cause project delays (11:47-48). Included in this category of "media control" is:

1. Storage and protection of card decks, tapes, disks, etc.
2. Media duplication and verification procedures.
3. Media conversion materials and procedures.
4. Media identification and level of revision systems.

Another subcategory of library control deals with "documentation control." It would be self-defeating to ensure the correctness and completeness of the requirements and design documentation if that documentation could be modified without proper control. Few software projects proceed to a successful conclusion without some changes to the original requirements and design. Because of this, procedures for the orderly and controlled insertion of changes must be defined, documented, and followed. By documenting every change, no matter how minor, the requirements document

reflects what the final product will be during all phases of development. This gives the quality assurance group a continuous and updated point of reference against which actual software behavior and structure can be compared to determine correctness (46:497).

Software Documentation. Perhaps the weakest link in software development is documentation. There are a number of sources of information which provide guidelines concerning software documentation, such as MIL-S-52779A and IEEE Standard 730. The military specification calls for referencing in the quality assurance plan all documentation standards and programming conventions and practices utilized on the software project (17:3). The IEEE standard calls for identification of the documentation governing the development and verification of the software and an explanation of how the documents are to be checked for adequacy (12:47). Software quality assurance must realize the requirement for good documentation and take steps to ensure that documentation which accompanies the developed software is complete, clear, and accurate.

An effective mechanism to ensure complete documentation accompanies the developed software is by using a Unit Development Folder (UDF) methodology. This documentation methodology consists of a notebook with a table of contents and formal schedule for entering information into the notebook for each major software unit (6). The ultimate objectives that the

content and format of the Unit Development Folder must satisfy are:

1. Provide an orderly and consistent approach in the development of the units of a program or project.
2. Provide a uniform and visible collection point for all unit documentation and code.
3. Aid individual discipline in the establishment and attainment of scheduled milestones.
4. Provide low-level management visibility and control over development process (33:251).

Figure 14 illustrates the role of the Unit Development Folder in the total software development process.

The Unit Development Folder is a readily accessible repository for all important documentation and notes created during the software development process, and becomes part of the final documentation package when the software project is completed (6). The following is a sample outline for a Unit Development Folder:

1. Cover Sheet
2. Schedules and Milestones
3. Requirements
4. Design Description
    a. Preliminary Design
    b. Code-to Design
    c. As-Built Design
    d. Interface Considerations

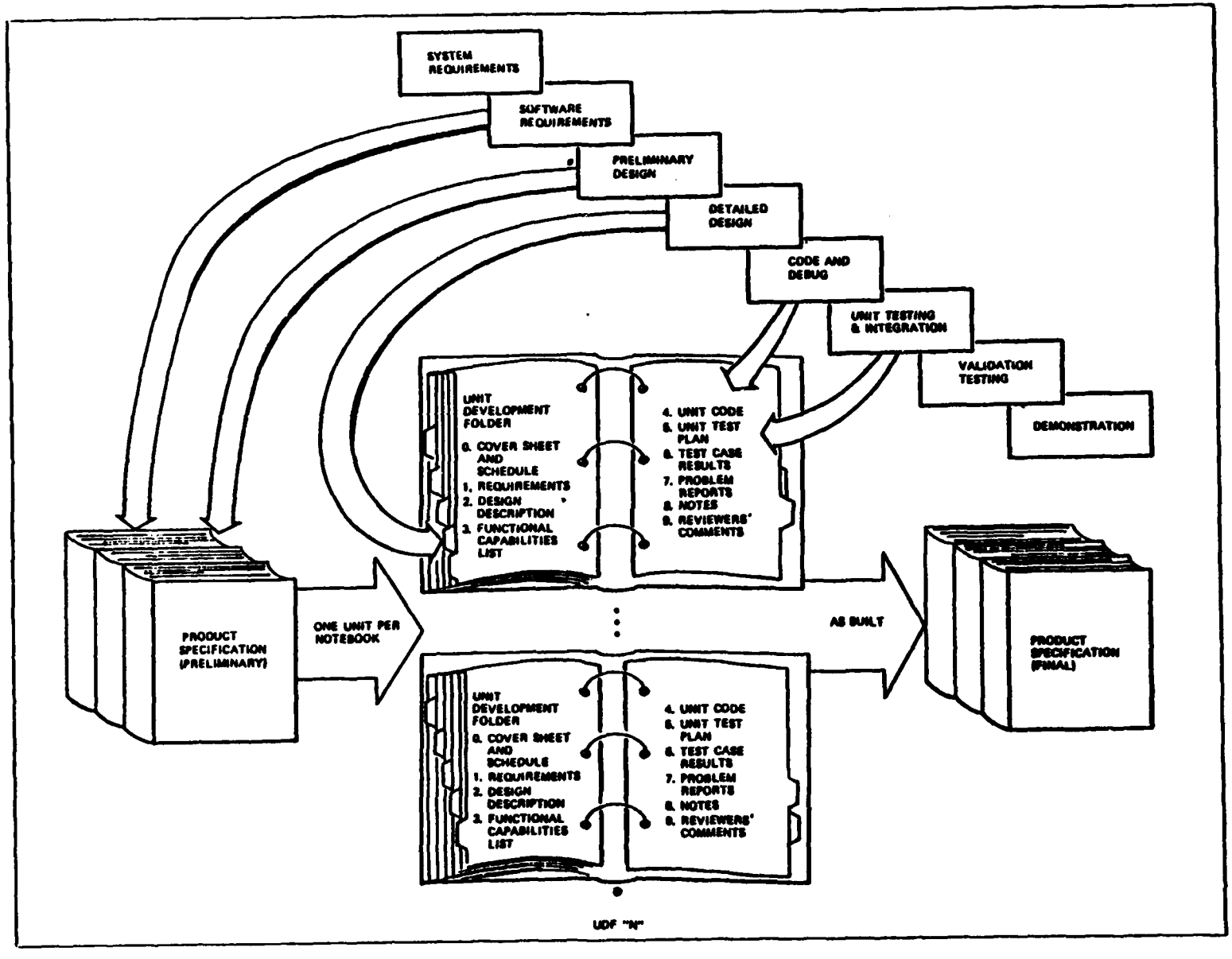



Figure 14. The UDF in the Development Process (33:252)

5. User Instructions
6. Unit Code Listing
7. Unit Test Plan
8. Test Results
9. Notes
10. Reviewer's Comments

Quality Assurance Techniques and Tools. Experience has indicated that good techniques and tools can serve as powerful aids in design, development, test, and maintenance of software (44:52). The difference between techniques and tools is very clear. Techniques consist of procedures arranged to simplify the evaluation process. Tools, on the other hand, are defined as automated aids used in evaluation of the developer's software or procedures (43:210).

Software quality assurance employ the methods of inspection, analysis, demonstration, and test. Inspection confirms compliance with stated requirements by examination. Analysis studies in detail to confirm an answer or result analytically. Demonstration provides tangible and visible evidence of compliance for review and comparison against stated objectives (43:210). Finally, tests are performed to find errors.

Quality assurance techniques and tools can be classified based upon the method they support. Table 4 list available techniques and tools by category. Of course this list is not all inclusive. For ease of further discussion, a glossary defining each technique and tool is provided in Appendix B. Also by using tables, each technique and tool is related with

usual quality assurance functions; and how each technique and tool supports the evaluation of software quality is illustrated.

First, Table 5 shows which technique supports various quality assurance functions. Next, Table 6 displays the degree of support each technique provides for assessing software quality factors. Third, Table 7 shows which tool supports what quality assurance function. Lastly, Table 8 displays the degree of support each tool provides for assessing software quality factors.

As the tables indicate, there are many techniques and tools available to improve software quality. One of the major tasks in developing an acceptable software quality assurance program is to select those techniques and tools in the most efficient and cost-effective manner (43:226). Selection should be done only after careful analysis of the objectives desired by the techniques and tools, and analysis of the criticality of the function to be evaluated. As an aid, Table 9 provides a checklist method for analysis.

Techniques and tools can be a valuable and useful addition to a software quality assurance program. But, they should be well documented and thoroughly tested. If the technique or tool itself is not validated, how can it be used with confidence to validate other software or to enforce standards (56:149).

TABLE 4

Quality Assurance Techniques and Tools (43:211)

| *Class* | *Inspection* | *Analysis* | *Demonstration* | *Test* |
|---|---|---|---|---|
| Technique | Auditing<br>Code inspection<br>Design inspection<br>Reviewing | Analytical modeling<br>Correctness proof<br>Error-prone analysis<br>Execution analysis<br>Post-functional analysis<br>Simulation<br>Standardization<br>Static analysis | Functional testing<br>Walk-throughs | Algorithm evaluation test<br>Correctness proofs<br>Equivalence classes<br>*Functional testing*<br>Logical testing<br>Path testing<br>Simulation<br>Stress testing<br>Symbolic execution |
| Tool | Consistency checker<br>Editor<br>Requirements tracer<br>Standards analyzer | Accuracy study processor<br>Comparator<br>Consistency checker<br>Cross-referencers<br>Data base analyzer<br>Decision tables<br>Dynamic analyzer<br>Editor<br>Flowcharter<br>Hardware monitor<br>Interface checker<br>Interrupt analyzer<br>Logic analyzer<br>Simulators<br>Software monitor<br>Static analyzer<br>Structure analyzer<br>Timing analyzer | Dynamic simulator<br>Hardware monitor<br>Software monitor<br>Standards analyzer<br>Test bed | Automated test generator<br>Comparator<br>Debugger<br>Dynamic analyzer<br>Dynamic simulator<br>Flowcharter<br>Hardware monitor<br>Instruction trace<br>Simulators<br>Software monitor<br>Test drivers, scripts<br>Test-result processor |
| Support tool (Common) | Language processors<br>Libraries<br>MIS<br>Standards<br>Text editor | | | |

TABLE 5

Relationship of Techniques to Quality Assurance Functions (43:213)

| Techniques | Function | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *Quality planning* | *Work tasking* | *Configuration management* | *Testing* | *Corrective action* | *Library controls* | *Design standards* | *Documentation standards* | *Reviews and audits* | *Subcontractor controls* |
| 1. Algorithm evaluation test | | | | X | | | | | | |
| 2. Analytical modeling | | | | X | | | | | | |
| 3. Auditing | X | X | X | X | X | X | X | X | X | X |
| 4. Code inspection | | | | | | | X | X | X | X |
| 5. Correctness proof | | | | X | | | | | | |
| 6. Design inspection | | | | | | | X | X | X | X |
| 7. Error-prone analysis | | | | X | | | | | X | |
| 8. Equivalence classes | | | | X | | | | | | |
| 9. Execution analysis | | | | X | | | | | X | |
| 10. Functional testing | | | | X | | | | | | |
| 11. Logical testing | | | | X | | | | | | |
| 12. Path testing | | | | X | | | | | | |
| 13. Post-functional analysis | | | | X | | | | | X | |
| 14. Reviewing | X | X | X | X | X | X | X | X | X | X |
| 15. Simulations | | | | X | | | X | | X | |
| 16. Standardization | X | X | X | X | X | X | X | X | X | X |
| 17. Static analysis | | | | X | | | | | X | |
| 18. Stress testing | | | | X | | | | | | |
| 19. Symbolic execution | | | | X | | | | | | |
| 20. Walk-throughs | | | | | | | X | X | X | X |

TABLE 6

Technique Effectiveness in Assessing Quality(43:214)

| Techniques | Quality property: Correctness | Efficiency | Integrity | Maintainability | Modifiability | Portability | Reliability | Testability | Usability |
|---|---|---|---|---|---|---|---|---|---|
| 1. Algorithm evaluation test | M | M | L | L | L | L | H | M | M |
| 2. Analytical Modeling | M | M | L | L | L | L | H | M | L |
| 3. Auditing | H | M | M | H | H | M | H | H | H |
| 4. Code inspection | M | L | L | M | M | M | H | M | M |
| 5. Correctness proof | H | L | L | L | L | L | M | M | L |
| 6. Design inspection | H | H | H | H | H | H | H | H | H |
| 7. Error-prone analysis | M | H | L | L | L | L | M | M | L |
| 8. Equivalence classes | M | M | M | L | L | L | H | H | L |
| 9. Execution analysis | H | H | M | M | M | L | M | L | L |
| 10. Functional testing | H | H | H | M | M | M | M | H | L |
| 11. Logical testing | M | M | L | L | L | M | H | M | L |
| 12. Path testing | M | L | L | M | M | L | H | H | L |
| 13. Post-functional analysis | M | H | L | M | M | M | H | M | M |
| 14. Reviewing | H | M | M | H | H | M | H | H | H |
| 15. Simulation | H | H | M | M | M | H | M | M | H |
| 16. Standardization | H | H | H | H | H | H | H | H | H |
| 17. Static analysis | H | L | M | M | M | L | M | M | M |
| 18. Stress testing | M | H | M | M | M | L | H | M | L |
| 19. Symbolic execution | M | M | M | M | M | M | H | M | L |
| 20. Walk-throughs | M | M | M | M | M | M | H | M | M |

Legend: H – High effectiveness
M – Medium effectiveness
L – Low effectiveness

TABLE 7

Relationship of Tools to Quality Assurance Functions (43:218)

| Tool | Function | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Quality planning | Work tasking | Configuration management | Testing | Corrective action | Library controls | Design standards | Documentation standards | Reviews and audits | Subcontractor controls |
| 1. Accuracy study processor | | | | X | | | | | | |
| 2. Automated test generator | | | | X | | | | | | |
| 3. Comparator | | | X | X | | | | X | | |
| 4. Consistency checker | | | | | | | X | X | | |
| 5. Cross-reference | | | | X | | | | | | |
| 6. Data base analyzer | | | | X | | | | | | |
| 7. Debugger | | | | X | | | | | | |
| 8. Decision tables | | X | X | X | X | X | | | X | X |
| 9. Dynamic analyzer | | | | X | | | | | | |
| 10. Dynamic simulator | | | | X | | | | | | |
| 11. Editor | | | | X | | | | | | |
| 12. Flowcharter | | | | X | | | X | | | |
| 13. Hardware monitor | | | | X | | | | | | |
| 14. Instruction trace | | | | X | | | | | | |
| 15. Interface checker | | | | X | | | | | | |
| 16. Interrupt analyzer | | | | X | | | | | | |
| 17. Language processor | | | | X | | | X | | | |
| 18. Libraries | | | X | X | | | | | | X |
| 19. Logic analyzer | | | | X | | | | | | |
| 20. MIS | | X | X | X | X | X | | | | X |
| 21. Requirements tracer | | | X | X | X | | | | X | X |
| 22. Simulator | | | | X | | | X | | | |
| 23. Software monitor | | | | X | | | | | | |
| 24. Standards | X | X | X | X | X | X | X | X | X | X |
| 25. Standards analyzer | | | | X | | | X | X | | |
| 26. Static analyzer | | | | X | | | X | X | | |
| 27. Structure analyzer | | | | X | | | X | | | |
| 28. Test bed | | | | X | | | | | | |
| 29. Test drivers, scripts | | | | X | | | | | | |
| 30. Test-result processor | | | | X | | | | | | |
| 31. Text editor | X | X | X | X | X | X | X | X | X | X |
| 32. Timing analyzer | | | | X | | | | | | |

TABLE 8

Tool Effectiveness in Assessing Quality (43:219)

| Tools | Quality Property | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Correctness | Efficiency | Integrity | Maintainability | Modifiability | Portability | Reliability | Testability | Usability |
| 1. Accuracy study processor | M | L | L | L | L | L | H | L | L |
| 2. Automated test generator | M | L | L | L | L | L | H | H | L |
| 3. Comparator | L | L | L | L | L | M | M | L | L |
| 4. Consistency checker | H | L | L | L | L | L | H | M | L |
| 5. Cross-reference | M | L | L | M | M | L | M | L | L |
| 6. Data base analyzer | M | L | L | M | M | M | M | M | L |
| 7. Debugger | M | M | L | L | L | L | H | L | L |
| 8. Decision tables | M | L | M | L | L | L | M | M | L |
| 9. Dynamic analyzer | H | H | L | M | M | L | H | M | L |
| 10. Dynamic simulator | H | M | L | L | L | L | H | L | L |
| 11. Editor | M | L | L | M | M | L | M | M | L |
| 12. Flowcharter | H | L | L | M | M | M | M | H | H |
| 13. Hardware monitor | M | H | L | L | L | L | H | M | L |
| 14. Instruction trace | L | M | L | L | L | L | H | M | L |
| 15. Interface checker | H | L | M | M | M | L | H | L | L |
| 16. Interrupt analyzer | H | L | M | M | M | L | H | L | L |
| 17. Language processor | H | M | M | H | H | H | H | H | H |
| 18. Libraries | L | L | H | L | L | L | L | L | H |
| 19. Logic analyzer | M | M | L | L | L | L | H | M | L |
| 20. MIS | H | L | L | L | L | L | L | L | L |
| 21. Requirements tracer | H | L | L | M | M | M | M | H | L |
| 22. Simulator | H | H | M | M | M | M | M | M | H |
| 23. Software monitor | M | H | L | L | L | L | H | M | L |
| 24. Standards | H | H | H | H | H | H | H | H | H |
| 25. Standards analyzer | H | L | M | H | H | M | M | M | M |
| 26. Static analyzer | H | L | M | M | M | L | M | M | M |
| 27. Structure analyzer | H | M | L | M | M | M | M | M | M |
| 28. Test bed | H | H | H | H | H | H | H | H | H |
| 29. Test drivers, scripts | M | L | L | L | L | L | H | H | L |
| 30. Test-results processor | M | L | L | M | M | M | M | H | H |
| 31. Text editor | M | L | L | H | H | L | L | H | H |
| 32. Timing analyzer | H | L | M | M | M | L | H | L | L |

Legend: H – High effectiveness
M – Medium effectiveness
L – Low effectiveness

TABLE 9

Factors Impacting Technique and Tool Selection (43:227)

| *Factor* | *Evaluation criteria* | *Yes* | *No* | *Notes* |
|---|---|---|---|---|
| Applicability | Is the tool or technique suited for the task?<br>If not, can it be modified easily to do the job? | | | |
| Availability | Is the tool or technique ready for use?<br>If so, can it be delivered to the customer in acceptable form if he desires it? | | | |
| Cost/Benefit | Can the use of the tool or technique be justified in terms of return on investment? If not, why use it? | | | |
| Experience | Has the tool or technique been used on other projects?<br>If so, how many and with what results?<br>If not, have adequate precautions been taken to manage the potential risk? | | | |
| Quality | Are the tools and techniques documented and is adequate user documentation available?<br>Are the tools under configuration control?<br>Have the tools been qualified formally?<br>Have the tools been developed using modern programming techniques and a High Order Language (HOL)? | | | |
| Resources | Have the potential sizing and timing growth implications of tool use been factored into the decision? | | | |
| Risk | Have the risks associated with developing or using the tool or technique been quantified? | | | |

## Chapter Summary

The preceding sections are the result of a comprehensive literature review, and present the current views on software quality assurance.

Primary problem areas that lead to schedule delays, cost overruns, or software products that fall short of their desired goals have been discussed. In addition, the components of an effective quality assurance program have been outlined and individually discussed.

In summary, the role of software quality assurance is to guide software development; however, everyone involved must participate in development if quality software is to be developed. With proper guidance, software can be developed that will satisfy user needs and garner the commitment of everyone involved because they had a part in developing it.

# III. Research Methodology

This chapter discusses the procedures used to collect and analyze available information in order to satisfy the research objectives proposed in Chapter I. Specifically, it focuses on data collection by means of a literature review and personal interviews, and criteria for data analysis.

## Data Collection

The first research objective deals with identifying various software quality methodologies. In order to satisfy this objective, an extensive literature review and personal interviews were performed.

Literature Review. The literature review provided an information baseline for further research endeavors, and consisted of the following sources:

1. Air Force Institute of Technology (AFIT) Libraries. In these libraries many computer and engineering journals were located. The most beneficial journals were those by the Institute of Electrical and Electronics Engineers (IEEE).
2. Defense Technical Information Center (DTIC). Extensive use was made of DTIC to gather information from all areas of the DoD relating to software quality.
3. National Aeronautics and Space Administration (NASA). The information obtained from the NASA search was, for the most part, too technical for use in this research.

But a few DTIC documents were located from the NASA search that were not identified by the DTIC search. This is difficult to explain since the same key words were used for both the DTIC and NASA searches.

4. Air Force Publications Library. The Wright-Patterson Air Force Base, Aeronautical Systems Division (ASD) publications library provided information from Air Force regulations. This proved very useful defining the Air Force position.
5. Military Standards and Specifications. The Wright-Patterson Air Force Base, Air Force Weapons Laboratory (AFWL) technical library contributed references from military standards and specifications used in this research effort.

Interviews. After considering the intent of the research effort, the level and nature of the data needed, and the availability of adequate respondents, a "personal interview" approach was chosen to complete the data collection. Usually the interview approach is the only practical way to gather opinions, intentions, or knowledge (26:213). Once the approach was selected, the communication mode was developed.

The communication mode involved a series of questions used as an interview guide. These questions were developed with the awareness that sequencing, wording, respondent sensitivity, and content influence the instrument development process (26:Chapt 8). In particular, question content had

the greatest impact on the process. To overcome this problem, the following questions were considered:

1. Should this question be asked?
2. Is the question of proper scope?
3. Can the respondent answer adequately?
4. Will the respondent answer adequately?

The interview questions were designed around the literature review from the previous chapter. Primarily, the section on the software quality assurance program made up the basic framework for the interview questions. Routine questions such as name, position, and experience were then added. A sample of the interview guide and cover letter is included in Appendix C.

Before any interviews can take place, the organizations involved with the interview process must be chosen. The criteria for selection included the following:

1. The organization must have an active quality assurance program in existence.
2. The organization must be involved in software development in some way.
3. The organization must be accessible to the researcher.
4. The organization must be willing to provide information.
5. There will be both civilian and Air Force organizations chosen.

Using the above criteria, the following civilian organizations were selected:

1. TRW Defense and Space Systems Group
2. SOFTECH Incorporated
3. NCR Corporation

In addition, the following Air Force organizations were chosen:

1. ASD/B1M Directorate of Projects
2. ASD/EN Computer Resources Focal Point
3. ASD/PMDQ Quality Assurance Division
4. ASD/YW Deputy for Simulators

After the organizations were chosen, a preliminary interview was scheduled with a senior manager from each organization. Actual interview respondents were selected with the advice of the aforementioned senior manager. The preliminary interview also servied as an opportunity to test and validate the interview questions. The interview guide and cover letter were distributed prior to the interview to allow preparation by the senior manager.

## Data Analysis

As the research objectives in Chapter I are subjective in nature, it follows that the research analysis will be primarily subjective. In addition, since the research covers a potentially large variety of concepts, each with its own uniqueness, the information does not lend itself to statistical analysis.

Criteria. Before any analysis can start, criteria must be established on which judgement may be based. After

reviewing the available literature, it was decided that the opinions represented in the literature was the most effective method of identifying the criteria needed for analysis. Those topic areas that recur most often in the literature were selected as the criteria needed for comparing and critiquing. To be more specific, the following criteria were chosen:

1. Planning. Planning is essential for the successful achievement of any project. It is critical that all planning be documented, such as in a quality assurance plan, so that knowledge will not be lost during the transition of personnel. Also, planning must start early in software acquisition and development. This allows early detection and correction of problems. Furthermore, higher management must be involved with the planning process, so that decisions can be made based on complete information. In addition, user involvement is imperative for complete requirements definition. In fact, user involvement is essential throughout the software development process to assure the highest quality product possible.
2. Staffing and Organization. The success of a quality assurance program begins with the personnel assigned to it. Members of the quality assurance staff should have a relatively high level of technical expertise and a thorough understanding of good software quality assurance practices. In addition, as a group they must report directly to the program or general manager. This allows for knowlegable

people to advise higher management on potential problems and possible solutions to those problems. Also, the software quality assurance staff must have direct authority over the software product by being an independent group within the organizational structure. This allows, as much as possible, unbias decisions to be made and unbias information to be gathered for analysis.

3. Reviews and Audits. Through a combination of reviews and audits control is maintained. Reviews and audits must be predefined, occur at key points in the development process, be understandable and complete, and are conducted in accordance with predefined standards. Again, a key point to mention is the importance of documenting the results of all reviews and audits. How else can new personnel or even management know what is going on unless the documentation is complete? Also by using reviews and audits, an organization's technical progress can be monitored through the software development phases.
4. Software Testing. Testing is a structured activity that occurs throughout software development. By increasing the awareness of software testing and encouraging test development, organizations can minimize problems with the software product and maximize software quality. It is crucial that independent verification and validation be used. Without it, confidence in the software product is jeopardized. Also, quality assurance personnel should be involved. This establishes a control that might be missing

if software developers did everything themselves. Finally the issue of documentation, without it the test results would be known to only a select few and management would be lacking important analysis information.

5. Library Control. Vital software library responsibilities include media control and documentation control. Protection of software media is necessary to prevent loss of information due to defective media or uncontrolled modification to the media. Likewise, documentation control ensures the correctness and completeness of the software documentation is maintained. Without these control measures, the integrity of the software product would be in question.

6. Documentation. As mentioned before, documentation is critical to understanding the software development process. Everyone must realize the requirement for up-to-date documentation and take steps to ensure that complete, clear, and accurate documentation accompanies the software product. Understanding what standards are available is essential in developing good documentation. Therefore, referencing all the necessary standards in one document is an invaluable aid to the software developer. Another aid to ensure complete documentation is the Unit Development Folder methodology. This methodology provides a collection point for all unit documentation and software code.

7. Techniques and Tools. Good techniques and tools can serve as powerful aids in all the software development phases. But they should be well structured, documented, and thoroughly tested. If not, how can they be used with certainty to validate other software?
8. Training. As previously mentioned, quality assurance personnel should have a thorough knowledge of quality assurance practices. An effective way of developing this knowledge is through training programs. Two such programs where effort should be focused are initial training and on-going training. An initial training program will allow new personnel to acquire the "bare essentials." Whereas an on-going training program will benefit everyone by keeping them educated in the most current practices used today. Anyway you look at it, quality assurance cannot grow or develop if there are not trained personnel in the field of quality assurance.

To have an effective quality assurance program, an organization must satisfy all the criteria mentioned above. This criteria establishes the minimum requirements and does not limit an organization from increasing its quality assurance program. To summarize the criteria discussed in this section, the following list is provided:

1. Planning.
    a. Planning is documented.
    b. High management is involved.

c. User involved.

d. Started early in software development and acquisition.

2. Staffing and Organization.

   a. Technical personnel included that are trained in quality assurance practices.

   b. Reporting is directly to program or general manager.

   c. Independent staff with authority over software product.

3. Reviews and Audits.

   a. Reviews and audits occur at pre-defined points.

   b. Results are well documented.

4. Software Testing.

   a. Independent verification and validation is used.

   b. Results are documented.

   c. Quality personnel are involved.

5. Library Control.

   a. Controlled access is essential for software media.

   b. Documentation changes are controlled.

6. Documentation.

   a. A Unit Development Folder approach is used.

   b. All software standards, specifications and such are referenced in one document.

7. Techniques and Tools.

   a. Techniques and tools used are validated.

   b. Techniques and tools are documented.

8. Training.

   a. On-going training is conducted.

   b. An initial training program is established.

Comparison. The second research objective deals with comparing software quality methods used by civilian and Air Force organizations. This objective was answered by comparing the civilian interview data and the Air Force interview data against the literature review data. All comparisons were accomplished using the criteria previously mentioned and deviations from the criteria were summarized.

Critique. The final research objective pertains to critiquing the effectiveness of the software quality methods. To complete this objective, all data collected was critiqued using the criteria mentioned before and areas needing improvement were identified. Next, recommendations and conclusions were developed to offer suggested ways to strengthen the quality assurance program.

## Chapter Summary

The contents of this chapter established the methodology used to collect and analyze the research data, and consisted of the following:

1. Collecting data on current quality assurance practices by means of:
    a. A literature review.
    b. A series of structured interviews.
2. Analyzing the data by:
    a. Comparing the civilian, Air Force, and literature review data.

b. Critiquing the collective data for effective software quality methods.

In summary, a well established research methodology is a road map to successful research by telling us were we went and how we got there.

# IV. Research Observations

This chapter will present the results of the comparison made between the civilian interview data, the Air Force interview data, and the literature review data. To prevent the possibility of compromising proprietary information, direct reference to specific organizations will not be made. Instead, summary references will be limited to "civilian companies" or "Air Force agencies."

## Organization

Organizations for software quality vary as do the organizational titles. Titles range from Quality Control to Quality Assurance to Product Assurance and others.

There appears to be a trend in civilian industry towards combining many of the functional disciplines into the same organization to take advantage of their related influences on software quality. This combining of disciplines provides a much better use of resources since the same individual could perform several related tasks that were previously fragmented. The Air Force agencies, for the most part, were fragmented into different functional disciplines.

This fragmentation contributes to the lack of a strong voice in making software decisions. In addition, it hinders the development of a unified quality assurance position that would maximize software benefits.

In industry, where top management felt software quality was important, the quality organization reported directly to the top management official. In U.S. defense contractor organizations, quality assurance is independent of manufacturing and reports directly to the top management official. Whereas quality assurance in the Air Force agencies was tied to manufacturing.

Even among the military services, the quality assurance organizations and the level in the organizational structure vary. In the Naval Material Command (NAVMAT), as illustrated in Figure 15, the Deputy Chief of NAVMAT for reliability, maintainability and quality reports directly to the NAVMAT Commander. Each of the Naval Systems Commands below NAVMAT also have a quality organization. At that level, a matrix concept is used (42:21).

The Army also has a strong and disciplined organization for quality assurance. As indicated in Figure 16, this organization reports to the Commander of the Development and Readiness Command (DARCOM). Each subordinate product command has a quality assurance organization for development and another for readiness. Quality assurance in DARCOM is organized to assure and assess quality at all phases of the acquisition process (42:22).

In contrast, the Air Force agencies had small quality assurance staffs. These staffs were located three levels below the Commander and reported to the Depute for Contracting and Manufacturing.

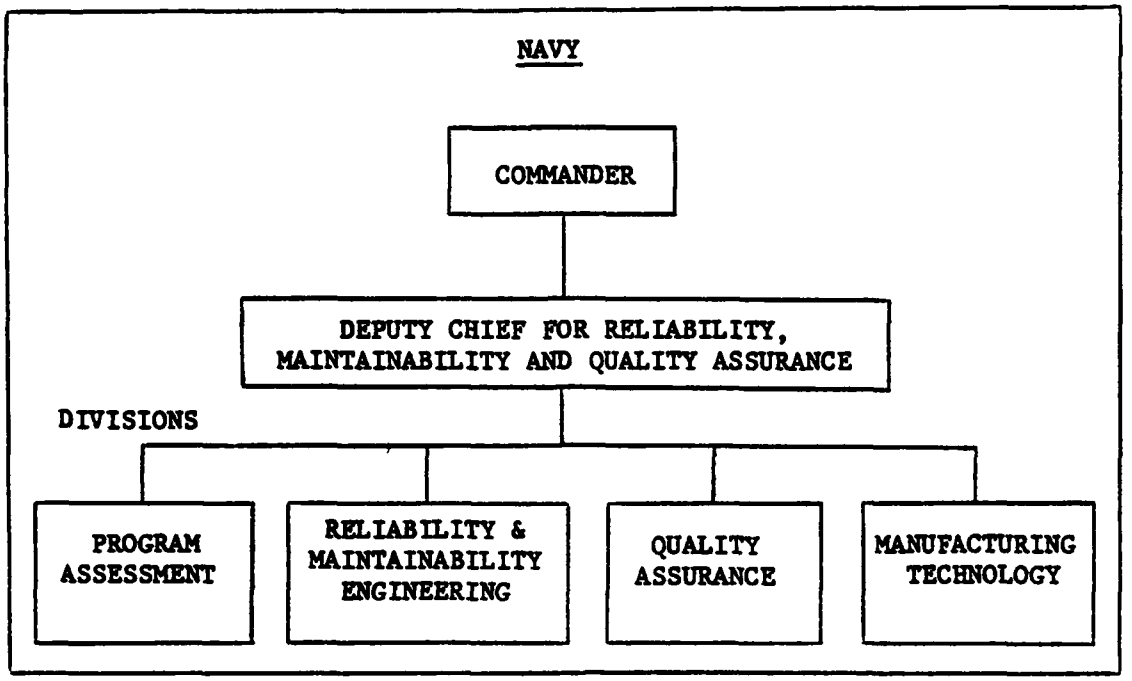



Figure 15. NAVMAT Organizational Structure (42:21)

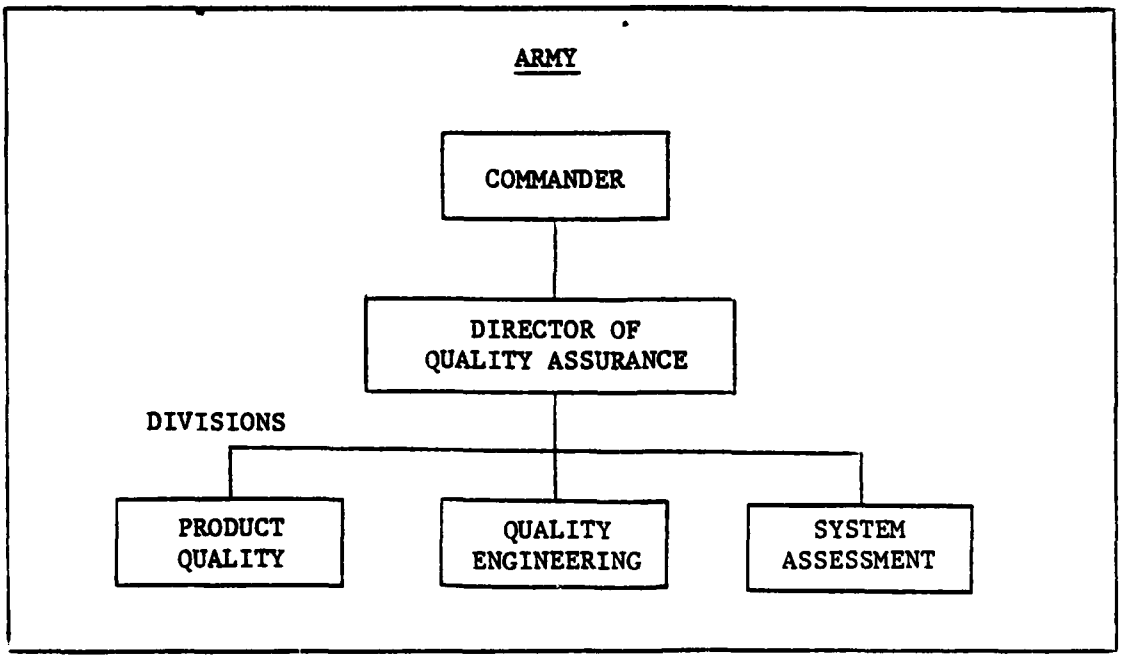



Figure 16. DARCOM Organizational Structure (42:22)

There appears to be a direct correlation between the influence of the quality assurance organization on management decisions and the level in the overall organization.

Quality assurance in the Army has an independent and equal voice with other functional organizations. Within the Air Force Agencies, the quality assurance organizations are normally too low in the overall organization to be influential. Because of this, quality assurance has neither an equal nor independent voice.

Planning

It was observed that quality assurance planning, within the civilian companies, included developing design requirements and criteria which are often published in company handbooks or procedure manuals which supplement industry or government standards. These efforts relect the experiences, lessons learned, and proven techniques for assuring a quality product.

The Air Force agencies generally did not have as disciplined an approach to assuring quality. The primary control document used was MIL-S-52779A, Software Quality Assurance Program Requirements. The intent of MIL-S-52779A is not to specify a particualr way for a company to organize. Instead, the purpose is to specify important software development functions that must be accomplished by contractors in order to assure quality (6). With this approach, an inherent weakness becomes obvious, just because functions are specified

doesn't mean that the contractor has the ability to assure the quality of the software product. An early review of the contractor's quality plan is needed.

Quality Measurement

Measurement of quality begins by determining the contractor's capability to produce the software product and the effectiveness of his quality assurance program to assure a quality product. The civilian companies surveyed evaluate a sub-contractor's total capability for producing and controlling conformance to the requirements. The sub-contractor's past performance is also considered as a strong indication of future performance. In addition, there is a tendency to select the best performers even though they may not be the lowest in development cost.

In the past, Air Force evaluations of a contractor's capability, quality assurance system, and quality management were often performed only by evaluating the contractor's Quality Assurance Program Plan during source selection. Often the influence to incorporate needed changes to the contractor's system is lost because these changes are not detected early in the competitive environment. After contract award, such changes are difficult to implement even though the contractor's system is obviously deficient and the change will result in improved software quality.

Currently, a review of the bidder's software development capability and capacity has been defined and used by

some Army and Air Force acquisition organizations. The Software Development Capability/Capacity Review (SDC/CR) covers software project management, management tools, development tools, and personnel (4).

Within 30 days of receipt of proposal, the source selection organization will schedule a SDC/CR to be held at the primary software development contractor's facility. All contractors who will have substantial software development responsibilities for the proposed system should be available for the review. The review will be limited to two working days and will take place no later than 60 days after the beginning of source selection. The SDC/CR is an integral part of the source selection process and is included in the evaluation for contract award (3:1).

User Involvement

The Air Force agencies agreed as to the extent of user involvement needed during the software development process. The user should be involved throughout the process, especially early in the planning phases. But the civilian companies had differing viewpoints. One viewpoint was similar to the Air Force agencies, since both groups are affected by the same regulations and standards.

Another viewpoint was from those companies not involved with DoD software contracts. They believe since the user is a source of new requirements, the user should be involved in writing the functional specifications. But users are not

directly involved after the specification has been written. This permits more management control over the development effort. Instead, users are allowed to submit changes. These changes must go through a control board for review. This allows management to control and coordinate the changes.

Even though there are varied positions on user involvement, one point must be clarified. The Air Force pays for its user involvement through increased software contract bids.

Testing

Whenever specific quality levels are required, verification testing is considered almost sacred. Only by such testing can management have confidence that the software will perform as intended. Testing is an iterative process and seldom if ever will the first time through be successful. Civilian companies have found out that numerous field failures are the direct result of failing to perform test and evaluation adequately.

In contrast, the Air Force tends to push state-of-the-art advances because of operational requirements. As a result, problems may arise in perfecting these new techniques and cause schedule delays and cost impacts which often result in cancelling testing that could have identified these problems. The irony is that when these inherent problems are not identified and eliminated early, then schedule and cost impacts tend to be even greater. Such schedule anc cost impacts further encourage shortcuts and the introduction of even more

problems, and the circle continues. Therefore, the more a new software product advances technology and performance, the greater the need for the application of quality assurance principles and techniques. Yet, the more likely they will not be used due to cost and schedule considerations.

Documentation

There are many problems or shortcomings connected with software quality documentation. To begin with, many regulations and standards used for software development are based on procedures developed for hardware. Also, references made to other regulations and standards are extensive and cross referencing can be considerably time consuming. Finally, there is a problem associated with too many regulations, standards, and guidelines. If personnel are flooded with directives, there is confusion regarding which to follow and the tendency may be to ignore the directives and improvise.

Therefore, to alleviate these problems, a consolidation of relevant information from all necessary regulations into one general software quality document will provide distinct advantages. First, a standardized plan will enable DoD to develop a standardized strategy with regard to software quality assurance. Next, DoD and contractors will become accustomed to standard operating procedures for evaluating the quality of software. And finally, control on the part of DoD will be more visible.

The groundwork in this direction was undertaken at the Joint Logistics Commanders (JLC) software workshop held in 1979. Recommendations and plans for implementation are developed, but no final document has been coordinated between the services.

Techniques and Tools

Neither the civilian companies nor the Air Force agencies extensively used software techniques and tools. The civilian companies developed their own tools to fit the software situation. Very few commercially produced tools were used. A noted weakness in this approach was the lack of formal tool validation and documentation. Only by repeated use were any defects in the tools corrected.

Within the Air Force agencies only a few tools were used, especially in simulation. These tools were purchased commercially and due to the competitiveness of the market were verified and validated by the producer. Probably the most effective technique the Air Force agencies used was the checklist approach to evaluate procedures.

Even though software tools abound, their use should only be considered where it will prove to be more cost effective and more accurate to have the task automated rather than performing it manually.

Training

Training and training programs in the U.S. military services and agencies range from extensive to almost non-

existent. The Defense Contracting Administration Service (DCAS) has two excellent quality assurance training programs. One is an individual certification program whereby quality assurance specialists are certified in one or more areas. The second DCAS training program is a formal intern program. This program is three years in length and consists of both classroom and on-the-job training (42:36).

On-the-job training is received by personnel to assist them to develop the skills needed to perform their assigned tasks. Both the civilian companies and the Air Force agencies relied heavily on this method of training. In addition, their software quality assurance training has primarily centered on reading any available guidebooks.

The need for training cannot be over emphasized. If software quality assurance personnel do not ahve the technical expertise to ensure that requirements are met, the resulting effects on the software product may be devastating. But if adequate training is provided, many benefits can be seen:

1. Training in the use of software tools and techniques will enable personnel to perform software quality assurance functions as a group separate from and independent of any other group.
2. Trained software quality personnel can perform activities with skill and confidence, and ensure that projects are completed within time and cost constraints.

3. Training programs strengthen the professional level of the personnel.

Benefits Gained

A number of benefits gained through the implementation of a quality assurance program were gathered while conducting the interviews. The following list is a composite of those benefits:

1. It has provided increased management visibility into the development process through reviews and audits.
2. Project risk has been reduced through more disciplined and thorough testing.
3. Quality assurance records have been centralized. These include problem reports, deviations and waivers, reviews and audits, and test and inspection reports among others.
4. It assures certain elements of quality in every phase of the software development.
5. The substantial reduction in the amount of rework has lead to a significant savings in life cycle costs.
6. It allows delivery of computer software which meets all contractual requirements.

Chapter Summary

In developing this chapter, a comparison was performed between civilian, Air Force, and the current literature on software quality assurance. Using the criteria established in Chapter III, a comprehensive account of the different

viewpoints was presented in preparation for the next chapter, where the recommendations and conclusions are offered.

Throughout the data collection ahd the analysis, one theme kept surfacing over and over. Software quality assurance, in itself, does not create quality in a product, instead it becomes a necessary part of everybody's job to ensure the user gets the highest quality product possible.

# V. Recommendations and Conclusions

This fifth and final chapter is the culmination of the research effort. It endeavors to present inferences and recommendations warranted by the nature and depth of the research. In addition, this research purports not to provide the entire solution to the software problem as stated in the first chapter, but to provide a contribution to the ultimate solution.

## Research Summary

Whether software is developed internally or by a contractor, the program or general manager must ensure that the software and its related documentation are of the highest quality possible. The most effective means of achieving these goals is a comprehensive quality assurance program. Research has indicated that the minimum key elements for a successful and effective program are planning, staffing and organization, reviews and audits, testing, library control, techniques and tools, documentation, configuration management, and training. In addition, it was observed that there was a common theme to the successful programs, both civilian and military. That theme was there must be a disciplined approach to implementing an effective quality assurance program.

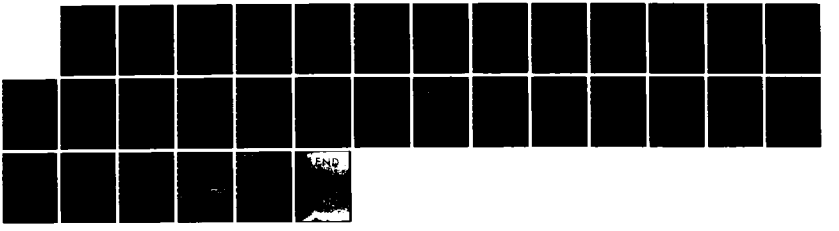
This section is grouped into categories in order to provide for a logical presentation. These categories include policy, organization, and education/training.

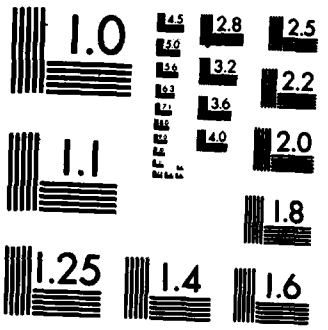
Policy. Although military and civilian teamwork along with excellent management have resulted in a few highly successful software projects, these effective management practices must be incorporated and used on a broader basis. The implementation of current guidance in DoD directives and military regulations (e.g., MIL-STD-1521A, MIL-S-52779A, MIL-Q-9858A, AFR 74-1) is often incomplete or not effective. In order to ensure proper emphasis is given to quality assurance, a unique service-wide regulation that integrates the tasks and functions defined in current regulations, standards, etc. should be developed and approved. This quality assurance regulation will not only provide for more effective use of scarce software resources, but its approach will aid in minimizing risks and achieve the required performance at the lowest possible cost.

Included in this regulation should be sections explaining each element of a software quality assurance program. The weakness in current regulations is that only select elements, such as quality assurance plans, are discussed. Without adequate consideration given to each element of software quality assurance, there cannot be an adequate quality assurance program developed. Once this objective has been obtained, then the work of implementing software quality assurance practices throughout the software development process will be performed with greater facility.

Organization. The recent trend has been to combine many of the functional disciplines into one organization, reporting

AD-A147 552 A SURVEY AND EVALUATION OF SOFTWARE QUALITY ASSURANCE 2/2
(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH
SCHOOL OF SYSTEMS AND LOGISTICS S P LAMB SEP 84
UNCLASSIFIED AFIT/GSM/LSY/84S-19 F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1

to top management. The Army (DARCOM) and the Navy (NAVMAT) both have this type of structure. There is no quantitative data to prove that these organizational changes have resulted in improved product quality; but increased attention to quality assurance is definitely a by-product of these changes.

The organizational changes will also eliminate the current Air Force fragmentation which exists. In addition, the following advantages will result from the changes:

1. It enhances front-end involvement during design and development phases.
2. It combines similar functions which are interrelated.
3. It provides clear and direct lines of communication.
4. It reduces duplication of efforts, multiple interpretations, and conflicting direction.
5. It provides for continuing visibility and attention to quality assurance by top management.

Education/Training. In comparing training available within the Air Force to that available to the Defense Contracting Administration Service (DCAS), it was obvious that the Air Force agencies interviewed lag far behind. DCAS has two well established programs which concentrate on quality assurance and the related disciplines. The Air Force agencies have virtually no formal training programs. As a result, there is an obvious requirement for education and training to assure that the work force remains current in quality assurance practices.

To begin with, all levels of management must have a strong appreciation for the benefits to be gained from a quality assurance program and they must continuously support the program if the full value of the benefits is to be derived. To achieve this goal, a short, intensive, on-site training program presented by an authority in quality assurance should be offered to top management, and should be under way at all times.

In line with raising management appreciation for quality assurance, there is a need to implement a formal training program to upgrade the skills and capabilities of the quality assurance work force. This formal approach can take the form of a certification program. There are advantages to certification programs. First, they will provide a level of depth at the technical level that will give the quality assurance personnel the expertise and confidence necessary to deal with contractor personnel. Next, they will be geared to government systems and provide standard training. Finally, certification programs will encourage further in-depth training.

Another significant area which needs to be addressed is the establishment of orientation instruction. This type of instruction is necessary regardless of the amount of training or technical expertise obtained by personnel. Orientation instruction will provide personnel with a brief description of the system, the specifications applicable to the system, and the work already accomplished.

Finally, there should be an Air Force Institute of Technology (AFIT) Education with Industry program established. This will provide access to "corporate" knowledge and experience not obtained in the Air Force due to its personnel rotation policy.

The education and training programs described above are needed to upgrade the skills of personnel involved with software quality assurance. These programs will also provide the basis for Air Force career development.

Recommendations

Based on the research performed in this thesis effort, the following recommendations are offered in hopes of enhancing the quality assurance process:

1. Combine disciplines in one organization reporting to top management and having line function responsibilities, not staff function responsibilities. It is important to emphasize that line managers are the focal points in management control. They are persons who judgement are incorporated in the approved plans, and they are the persons who must influence others and whose performance is measured. Staff people collect, summarize, and present information that is useful in the process. However, the significant decisions are made by the line managers, not by the staff (2:21).
2. Top management must continue to re-emphasize the importance of quality assurance within the organization and

display a positive philosophy of commitment to the quality process. As has been discussed, without top management acceptance, quality assurance programs produce a less than desirable output.

3. Education and training programs must be developed to assure availability of qualified personnel. These programs should include the following:
   a. Intensive top management seminars.
   b. Certification programs.
   c. Orientation instruction.
   d. Education with Industry programs.
4. A unique software quality assurance document should be developed for joint service application. As a minimum, the elements of a quality assurance program should be addressed. While extensive progress has been made by the Joint Logistic Commanders (JLC), much work needs to be completed before a single document finds its way into use.
5. Clear and concise documentation is an on-going problem at any software developemnt facility. All efforts should be made to review documentation and assist where possible to improve any deficiencies.
6. Techniques and tools, whether uniquely developed or purchased commercially, should be verified and validated as thoroughly as the actual software product. The rationale of repeated usage cannot be accepted as validation. All

aspects of the technique or tool should be documented in order to develop confidence in the software product.

7. The philosophy of Software Development Capability/Capacity Reviews (SDC/CR) should be incorporated into software development contracts. So far, this review has proven successful in evaluating a contractor's software capability prior to source selection. As a result, many potential problems can be prevented when inadequate software contractors are eliminated early.

## Problems Encountered

As with any research effort of this size, problems occurred that should be mentioned to help explain any shortcomings. To begin with, the amount of time and effort needed to complete the literature review took away from the time available to do the field work and subsequent analysis. In addition, proprietary information was encountered while conducting the interviews. Due to the sensitive nature of this information, no direct reference was made to any specific company or organization.

## Further Research

In this section, two major areas for further research are discussed. These are quality assurance and training.

Quality Assurance. As this research has indicated, software quality assurance is an area where software developers must place a great deal of emphasis. It is felt that

another approach should be undertaken to further study software quality assurance. One way research could be conducted is by sending questionnaires to personnel in quality assurance departments at other companies in the software industry. The questionnaires should be designed to determine how software quality assurance is managed. From this, the Air Force would be able to evaluate the different methods and develop quality procedures for the software the Air Force purchases.

Training. Technological advances in software development are resulting in improved quality assurance practices. This is creating the need for software quality assurance personnel with more specialized skills. Therefore, training should be a major goal of any quality assurance organization. This research effort has found that formal training is often neglected. Further study should be conducted as to the benefits and possible cost savings that could be derived from quality assurance training and education. In addition, types of training programs should be discussed so as to offer the widest spectrum of training possible. Also, different Air Force organizations involved with software development should be researched to determine the requirements for training programs in software quality assurance.

## Conclusion

It is critical to the success of large weapons systems that software be delivered to the field with the minimum

number of errors. For this reason, the discipline of software quality assurance is needed.

To provide the highest level of software quality, the entire development process must include quality checks at each step from design through acceptance test. An active software quality assurance program that identifies and corrects errors during the development process is necessary. This effort will lead to significant defects being identified and resolved early; defects that would normally lead to major schedule and cost impacts on the development effort.

If the quality of software is to improve, greater emphasis must be placed on software quality assurance as a separate discipline. Quality software cannot be attained by following hardware oriented plans and procedures, such as MIL-Q-9858A. Therefore, software conformance standards must be provided.

Technology is constantly changing and advancing, and provision must be made to update personnel in the state-of-the-art quality assurance practices. Continual training is essential, both for those personnel who have quality assurance background and those who do not.

Software quality assurance "is expected to employ the best possible techniques and practices, assuring that the products delivered as weapons systems are the best that modern software technology can produce" (23:118). There is no room for anything less.

## Appendix A: Directives/Mil-Standards/Regulations

This appendix identifies the government documents, by type, that impact the quality assurance effort, and is provided to supplement the research.

### Department of Defense Directives/Instructions

| | |
|---|---|
| DODI 4105.65 | Acquisition of Automation Data Processing Computer Program and Related Services |
| DODI 5000.31 | Interim List of DoD Approved High Order Programming Languages |
| DODI 5010.21 | Configuration Management Implementation Guidance |
| DODI 5010.27 | Management of Automated Data System Development |
| DODI 7041.3 | Economic Analysis and Program Evaluation for Resource Management |
| DODD 4105.55 | Selection and Acquisition of Automated Data Processing Resources |
| DODD 4120.21 | Specifications and Standards Application |
| DODD 4155.1 | Quality Assurance |
| DODD 4155.19 | NATO Quality Assurance |
| DODD 4160.19 | Department of Defense Automatic Data Processing Equipment Reutilization Program |
| DODD 5000.1 | Major System Acquisition |
| DODD 5000.2 | Major System Acquisition Process |
| DODD 5000.3 | Test and Evaluation |
| DODD 5000.19-L | Acquisition Management Systems and Data Requirements Control List (AMSDL) |
| DODD 5000.29 | Management of Computer Resources in Major Defense Systems |
| DODD 5010.19 | Configuration Management |

| | |
|---|---|
| DODD 5100.40 | Responsibility for the Administration of the DoD Automatic Data Processing Program |

Military Standards/Specifications

| | |
|---|---|
| MIL-STD-109B | Quality Assurance Terms and Definitions |
| MIL-STD-480 | Configuration Control-Engineering Changes, Deviations and Waivers |
| MIL-STD-481 | Configuration Control-Engineering Changes (Short Form) |
| MIL-STD-482 | Configuration Status Accounting Data Elements and Related Features |
| MIL-STD-483 | Configuration Management Practices for Systems, Equipment, Munitions and Computer Programs |
| MIL-STD-490 | Specification Practices |
| MIL-STD-499A | Engineering Management |
| MIL-STD-781 | Definitions of Effectiveness Terms for Reliability, Maintainability, Human Factors and Safety |
| MIL-STD-881A | Work Breakdown Structures for Defense Material Items |
| MIL-STD-882 | System Safety Program for Systems and Associated Subsystems and Equipment |
| MIL-STD-1521A | Technical Reviews and Audits for Systems, Equipment, and Computer Programs |
| MIL-STD-1535A | Supplier Quality Assurance Program Requirements |
| MIL-STD-1553B | Aircraft Internal Time Division Command/ Response Multiplex Data Bus |
| MIL-STD-1588 | JOVIAL (J3) |
| MIL-STD-1589C | JOVIAL (J73) |
| MIL-STD-1679 | Weapon System Software Development (NAVY) |
| MIL-STD-1750A | Sixteen-Bit Computer Instruction Set Architecture |

| | |
|---|---|
| MIL-STD-1760 | Aircraft/Stores Electrical Interconnect System |
| MIL-STD-1815 | Ada |
| MIL-STD-1862 | NEBULA 32 Bit Instruction Set Architecture |
| MIL-STD-SDS | Defense System Software Development (Preliminary Draft) |
| MIL-STD-SQAM | Software Quality Assessment and Measurement (Proposed) |
| MIL-Q-9858A | Quality Program Requirements |
| MIL-S-52779A | Quality Assurance Program Requirements |
| MIL-S-83490 | Specifications, Types and Forms |
| MIL-HDBK-334 | Evaluation of a Contractor's Software Quality Assurance Program |

Regulations/Manuals

| | |
|---|---|
| AFR 70-1 | Procurement of AF Assigned Items |
| AFR 70-18 | Local Purchase Program (AFSC Supplement) |
| AFR 76-15 | Procurement Quality Assurance |
| AFR 300-1 | Automatic Data Processing Program Management |
| AFR 300-2 | Management of Automatic Data Processing Systems |
| AFM 300-6 | Automatic Data Processing Resource Management |
| AFR 300-10 | Computer Programming Languages |
| AFM 300-12 | Procedures for Managing Automatic Data Processing Systems |
| AFR 310-1 | Management of Contractor Data |
| AFR 310-2 | Management of USAF ADP Program |
| AFR 800-14 | (Vol. I) Management of Computer Resources in Systems |

| | |
|---|---|
| AFR 800-14 | (Vol. II) Acquisition and Support Procedures for Computer Resources in Systems |
| AFM 172-1 | USAF Budget Manual |
| AFCMDP 800-2 | Acquisition Management - Contract Management Guide |
| AFCMDR 70-16 | Supporting Contract Administration |
| AFCMDR 70-24 | Subcontract Management Program |
| AFCMDR 74-1 | Procurement Quality Assurance Program |
| AFCMDR 84-1 | Production Manufacturing Operations |
| AFCMDR 178-1 | Contractor Management System Evaluation Program |
| AFCMDR 800-1 | Acquisition Management - Contract Management Engineering |
| AFCMDR 800-3 | Embedded Computer Resources |
| AFCMDR 800-11 | AFCMD Memorandum of Agreement Management System |
| AFSC Sup 1 | (AFR 300-10) Computer Programming Languages |
| AFLCR 800-21 | Management and Support Procedures for Computer Resources Used in Defense Systems |
| AFLCP/ AFSCP 800-34 | Acquisition Logistics Management |

## Appendix B: Glossary of Techniques and Tools

This appendix defines the techniques and tools employed in a quality assurance program. The following glossary is neither all inclusive nor totally complete (43:212-225).

### Technique Glossary

1. ALGORITHM EVALUATION TEST: A technique used to evaluate critical algorithm trade-offs (i.e., speed versus size versus precision) before the design is finalized. Often called "the hardest out first method", the technique creates a detailed design based upon trial coding results for key algorithms. The algorithms are often extensively exercised in a simulated environment to ensure mission requirements are satisfied.

2. ANALYTICAL MODELING: A technique used to express mathematically (usually by a set of equations) a representation of some real problem. Such models are valuable for abstracting the essence of the subject of inquiry. Because equations describing complex systems tend to be complicated and often impossible to formulate, it is often necessary to make simplifying assumptions, which may tend to distort accuracy.

3. AUDITING: A formal technique employed to examine and verify through inspection either the status of a program and its documentation or the adherence of project personnel to established procedures. Scheduled audits are normally contractually imposed and periodically held. Unscheduled audits are utilized at random intervals to assess compliance with quality requirements.

4. CODE INSPECTION: A disciplined technique used for inspecting the code and identifying errors. Participants have well-defined roles and criteria for evaluating the code. If errors are identified, the code is reworked. Follow-up procedures are used to ensure that the errors have been corrected.

5. CORRECTNESS PROOFS: A technique used to prove the correctness of programs using means similar to those employed to prove mathematical theorems. Axioms and theorems derived are used to establish the validity of the program with respect to a precise specification of its purpose.

6. DESIGN INSPECTION: A disciplined technique used for inspecting the design and identifying errors. Participants have well-defined roles and criteria for evaluating the design. If errors are identified, the design is reworked. Follow-up procedures are used to ensure the errors have been corrected.

7. ERROR-PRONE ANALYSIS: A technique employed during coding to identify areas of the program that have required abnormally frequent correction and change. These areas can either be reworked or subjected to an extensive test effort.

8. EQUIVALENCE CLASSES: A technique used to automatically identify a complete set of test cases for a program. The set is interpreted in terms of inequalities involving program variables that define a set of conditions necessary for the particular program flow to actually occur.

9. EXECUTION ANALYSIS: A technique employed during test to investigate program behavior errors and to identify areas in the code that were either untested or not fully tested. The program is executed and statistics are collected. Test results and the statistics are then analyzed to insure that each interface, functional and test requirement has been correctly mechanized by the code.

10. FUNCTIONAL TESTING: A technique used to demonstrate that the software performs it specifications satisfactorily under normal operating conditions, computing nominally correct output values from nominal input values.

11. LOGICAL TESTING: A technique used to confirm that the code performs its computation correctly. Items validated by logical testing include arithmetic (i.e., precision, accuracy, etc.), error handling, initialization, interfaces, and timing.

12. PATH TESTING: A technique used to confirm that certain test-effectiveness measures based on the program's control topology have been realized. The technique assures that a sufficient number of statements, branch paths, and subroutine calls have been exercised during program execution. It also helps identify a complete set of test cases for the program.

13. POST-FUNCTIONAL ANALYSIS: A technique employed after completion of functional testing to identify functionally weak areas in the program. The recorded test

results are analyzed and the quality of the final product is determined.

14. REVIEWING: A technique employed to examine and verify through inspection either the status of a program and its documentation or adherence of project personnel to established procedures. Scheduled reviews are normally contractually imposed and periodically held. Informal reviews are held frequently to assess in detail the technical adequacy of the software product.

15. SIMULATION: Simulation is the process of studying specific system characteristics by use of models exercised over a period of time and a variety of conditions for the purpose of evaluating alternatives, timing, system capacities, performance, and constraints within the confines of that system. Simulation can be used by quality assurance throughout the life cycle. It can assist in evaluating conceptual trade-offs. It can also be used to model the environment and provide realistic test inputs to a program being examined.

16. STANDARDIZATION: A technique used to create an authoritative model against which products and/or procedures can be compared in order to determine their quality. Software items for which standards can be easily established include documentation, languages, designs, and structured programming.

17. STATIC ANALYSIS: A technique employed during test to identify weaknesses in the source code. The syntax of a program is examined and statistics about it are generated. Items such as relationships between module, program structure, error-prone constructions, and symbol/subroutine cross-references are checked and violations of established rules are anlyzed.

18. STRESS TESTING: A technique employed to confirm that the code performs its specifications satisfactorily under extreme operating conditions, computing nominally correct output values from worst case input values (i.e., singularities, end points for the range of data, etc.)

19. SYMBOLIC EXECUTION: A technique that employs symbolic data to confirm that the software performs properly. Symbolic execution allows one to choose intermediate points in the test spectrum ranging from individual test runs to correctness proofs. Its results can be used to develop a minimum set of test cases.

20. WALK-THROUGHS: A technique used for reviewing the design or code and identifying errors. The responsible programmer discusses his product with his peers and solicits their constructive advice. Product modifications are then made at the discretion of the programmer to correct problems identified during review.

Tool Glossary

1. ACCURACY STUDY PROCESSOR: A computer program used to perform calculations or assist in determining if program variables are computed with required accuracy. The processor accepts mathematical equations and data as inputs. It then uses the data as variables in the equations and solves them.

2. AUTOMATED TEST GENERATOR: A computer program that accepts inputs specifying a test scenario in some special language, generates the exact computer inputs, and determines the expected results.

3. COMPARATOR: A computer program used to compare two versions of the same computer program under test to establish identical configuration or to specifically identify changes in the source coding between the two versions.

4. CONSISTENCY CHECKER: A computer program used to determine 1) if requirements and/or designs specified for computer programs are consistent with each other and their data base and 2) if they are complete.

5. CROSS-REFERENCE: A group of computer programs that provide cross-reference information on system components. For example, programs can be cross-referenced with other programs, macros, parameter names, etc. This capability is useful in problem-solving and testing to assess impact of changes to one area or another. This capability should be provided in most compiler environments.

6. DATA BASE ANALYZER: A computer program that reports information on every usage of data, identifies each program using any data elements, and indicates whether the program inputs, uses, modifies, or outputs the data element. Any unused data is printed. Errors dealing with misuse and nonuse of data and conflicts in data usage are identified during the analysis.

7. DEGUGGER: Compile and execution-time checkout and degug capabilities that help identify and isolate

program errors. They usually include commands or directives such as DUMP, TRACE, MODIFY CONTENTS, BREAK-POINT, etc. Some debuggers operate at the source level and others at the object level with some additional source information.

8. DECISION TABLES: A mechanism used to represent information on program conditions, rules, and actions in tabular form that can be automatically translated to executable code by a processor. Decision tables are a tabular representation of the design which can be used to clarify the control flow of decision alternatives by presenting the information in a concise and understandable format.

9. DYNAMIC ANALYZER: A computer program that instruments the source code by adding counters and other statistics gathering sensors and produces reports on how thoroughly the various portions of the code have been exercised after the augmented code is executed. Dynamic analyzers provide information useful for tuning, optimization, and test case design.

10. DYNAMIC SIMULATOR: A computer program used to check out a program in a simulated environment comparable to that in which it will reside. Closed-loop effects between computer and environmental models are gained when the various models respond to inputs and outputs. The simulator allows the environment to be stabilized at a specific configuration for any number of runs required to observe, diagnose, and resolve problems in the operational program.

11. EDITOR: A computer program used to analyze source programs for coding errors and to extract information that can be used for checking relationships between sections of code. The editor can scan source code and detect violations to specific programming practices and standards, construct an extensive cross-reference list of all labels, variables and constants, and check for prescribed program formats.

12. FLOWCHARTER: A computer program used to show in detail the logical structure of a computer program. The flow is determined from the actual operations as specified by the executable instructions, not from comments. The flowcharts generated can be compared to flowcharts provided in the computer program design specification to show discrepancies and illuminate differences.

13. HARDWARE MONITOR: A unit that obtains signals from a host computer through probes attached directly to the

computer's circuitry. The signals obtained are fed to counters and timers and are recorded. These data are then reduced to provide information about system and/or program performance (CPU activity, channel utilization, etc.).

14. INSTRUCTION TRACE: A computer program used to record every instance a certain class of operations occurs and triggers event-driven data collection. In some cases, this creates a complete timed record of events occurring during program execution.

15. INTERFACE CHECKER: A computer program used to automatically check the range and limits of variables as well as the scaling of the source program to assure compliance with interface control documents.

16. INTERRUPT ANALYZER: A computer program that determines potential conflicts to a system as a result of the occurrence of an interrupt.

17. LANGUAGE PROCESSORS: Computer programs used to translate high-level or symbolic instruction mnemonics into computer-oriented code capable of being executed by a computer. Compilers, assemblers and meta-assemblers are example tools used for program development. Preprocessors have been developed to support implementation of modern programming techniques.

18. LIBRARIES: A collection of organized information used for reference or study. Many varieties of library systems can be implemented. Some manage the storage and distribution of the computer program in both source and object form. Others manage the computer program, its documentation and related test data.

19. LOGIC ANALYZER: A computer program use to automatically reconstruct equations forming the basis of a program and to flowchart assembly language programs.

20. MANAGEMENT INFORMATION SYSTEM: Consists of a computer based information system (a particular combination of human service, material service, and equipment service) for the purpose of gathering, organizing, communicating, and presenting information to be used by individuals for planning and controlling an enterprise.

21. REQUIREMENTS TRACER: A computer program used to provide traceability from requirements through design and implementation of the software products. Traceability is characterized to the extent that an audit trail exists for the successive implementation of each requirement.

22. SIMULATOR: A computer program that provides the target system with inputs or responses that resemble those that would have been provided by the process for the device being simulated. The simulator's function is to present data to the system at the correct time and in an acceptable format.

23. SOFTWARE MONITOR: A computer program that provides detailed statistics about system performance. Because software monitors reside in memory, they have access to all the tables the system maintains. Therefore, they can examine such things as core usage, queue lengths, and individual program operation to help measure performance.

24. STANDARDS: Procedures, rules, and conventions used for prescribing disciplined program development. Architecture and partitioning rules, documentation conventions, language conventions, configuration, and data management procedures, etc., are typical examples under this category.

25. STANDARDS ANALYZER: A computer program used to automatically determine whether prescribed programming standards and practices have been followed. The program can check for violations to standards set for such conventions as program size, commentary, structure, etc.

26. STATIC ANALYZER: A computer program used to provide information about the features of a source program. This type of tool examines the source code statically (not under execution conditions) and performs syntax analysis, structure checks, module interface checks, event sequence analysis and other similar functions.

27. STRUCTURE ANALYZER: A computer program used to examine source code and determine that structuring rules set for either the control or data structure, or both, have been obeyed.

28. TEST BED: A test site composed of actual hardware (hardware test site) or simulated equipment (software test site) or some combination. A hardware test site uses the actual computer and interface hardware to check out the hardware/software interfaces and actual input/output. The program execution is confirmed using actual hardware timing characteristics, but the output is limited and test repeatability is a problem. A software test site uses an instruction level and/or statement level simulator to model actual hardware. A software test site permits full control of inputs and computer

characteristics, allows processing of intermediate outputs without destroying simulated time, and allows full test repeatability and good diagnostics.

29. TEST DRIVERS, SCRIPTS: To run tests in a controlled manner, it is often necessary to work within the framework of a "scenario" -- a description of a dynamic situation. To accomplish this, the input data files for the system must be loaded with data values representing the test situation or events to yield recorded data to evaluate against expected results. These tools permit generation of data in external form to be entered into the system at the proper time.

30. TEST-RESULT PROCESSOR: A computer program used to perform test output data reduction, formatting, and printing. Some perform statistical analysis where the original data may be the output of a monitor.

31. TEXT EDITOR: A computer program used to prepare documentation and perform work-file edits (erase, insert, change, and move words or groups of words). The program requires a facility for on-line storage and recall of text units for inspection, editing, or printing.

32. TIMING ANALYZER: A computer program that monitors and prints execution time for all program elements (functions, routines, and subroutines).

# Appendix C: Sample Letter and Interview Guide

DEPARTMENT OF THE AIR FORCE
AIR FORCE INSTITUTE OF TECHNOLOGY (AU)
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433

REPLY TO ATTN OF: LS (Mr. Michael D. Bates, 255-4845)

SUBJECT: Software Quality Assurance

TO:

1. A master's degree candidate (Captain Steven P. Lamb) in the Systems Management Program at the Air Force Institute of Technology (AFIT) is conducting research under the guidance of the AFIT graduate faculty (Mr. Michael D. Bates). Captain Lamb has defined a research objective which will determine approaches to improving the quality of computer software before turnover to the Air Force. Based on this objective, he will be able to provide recommendations on the content of an effective quality assurance program.

2. To achieve the objective, Captain Lamb needs to interview people who know your organization's software quality assurance program. May we have your permission for Captain Lamb to interview the people you designate? In addition, would you send Captain Lamb any printed information, such as organizational charts, that support the interview guide Captain Lamb plans to use. Our contact point for this effort is Mr. Bates, AFIT/LSY, Wright-Patterson AFB, Ohio, 45433.

3. I would appreciate your cooperation in helping Captain Lamb complete this research project. Of course, we will amalgamate the responses so that none are attributable to an individual. Thank you for your help.

Larry L. Smith

LARRY L. SMITH, Colonel, USAF
Dean
School of Systems and Logistics

1 Atch
Interview Guide

AIR FORCE—A GREAT WAY OF LIFE

## Interview Guide

### Background

1. Name.
2. Position.
3. How many years have you worked here?
4. In what field is your formal training?

### Quality Assurance Planning

5. What planning exercises, documents and such are used to assure quality software prior to the start of a project?
6. What problems are encountered during the quality assurance planning process?
7. Who are the key people involved in this planning process?
8. Is user involvement in the software development process encouraged? If so, how?

### Staffing and Organization

9. Where does the quality assurance goup fit in the organization?
10. What type of authority does the quality assurance group have over the software product?
11. What education and training do the personnel in the quality assurance group have?
12. What functions are performed and/or controlled by the quality assurance group?
13. How many people are permanently assigned to the quality assurance group?

### Reviews and Audits

14. Is a schedule developed for conducting software reviews and audits during the development life cycle?

15. If so, when is this schedule developed and where is it documented?

16. Who performs the reviews and audits?

17. How are the results of the software reviews and audits documented and published?

Software Testing

18. Who performs the software testing throughout the development life cycle?

19. What is the organizational relationship between the quality assurance group and the testing group?

20. Who certifies that the test results are the actual findings of the tests?

21. What documents are used to control software testing?

Library Control

22. What safeguards assure no unauthorized changes are made to the developmental software?

23. What would be considered "controlled materials"?

Software Documentation

24. What standards are followed when preparing software documentation?

25. Is a Unit Development Folder (UDF) approach used to keep track of documentation for each software unit or module?

Quality Assurance Techniques and Tools

26. Techniques consist of procedures arranged to simplify the evaluation process (e.g., walk-throughs, audits, etc.). Are quality assurance techniques used?

27. Tools are automated aids used in evaluation (e.g., editors, simulators, etc.). Are quality assurance tools used?

28. Where are quality assurance techniques and tools identified?

29. Are quality assurance techniques and tools verified as to their usefulness prior to implementing?

Training

30. What type of training do new quality assurance personnel receive?

31. If a new software product is designed, how is the user trained?

Follow-up Considerations

32. Who handles problems with software after release?

33. What methods of feedback are used to obtain information from the user after release?

Conclusion

34. Is there anything you would like to add?

35. Are there any questions I can answer for you?

## Bibliography


1. Alberts, David S. "The Economics of Software Quality Assurance," Tutorial: Software Testing and Validation Techniques. 348-357. IEEE, Inc., New York, 1978.

2. Anthony, Robert N. and David W. Young. Management Control in Non-Profit Organizations. Homewood IL: Richard D. Irwin, Inc., 1984.

3. Aeronautical Systems Division. Software Development Capability/Capacity Review. Instructional manual. ASD/EN, Wright-Patterson AFB OH, May 1984.

4. Babel, Philip S. Joint Service Acquisition Management Initiatives. Report. ASD/EN, Wright-Patterson AFB OH, May 1984.

5. Baker, Emanuel R. and Matthew J. Fisher. "A Software Quality Framework," Concepts, 5: 95-107 (Autumn 1982).

6. Bates, Michael D. Lecture materials distributed by the Department of System Acquisition Management. School of Systems and Logistics, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, February 1984.

7. Blanchard, Benjamin S. Engineering Organization and Management. Englewood Cliffs NJ: Prentice-Hall, Inc., 1976.

8. Boebert, W. Earl. "Software Quality Through Software Management," Software Quality Management, edited by John D. Cooper and Matthew J. Fisher. New York: Petrocelli Books, Inc., 1979.

9. Boehm, Barry W. Software Engineering Economics. Englewood Cliffs NJ: Prentice-Hall, Inc., 1981.

10. -----. "Quantitative Evaluation of Software Quality," Tutorial: Models and Metrics for Software Management Engineering. 218-231. IEEE, Inc., New York, 1980.

11. Boeing Aerospace Company. Software Acquisition Engineering Guidebook: Software Quality Assurance. Report No. ASD-TR-78-47. ASD/ENE, Wright-Patterson AFB OH, January 1979 (AD-A082 425).

12. Buckley, Fletcher. "A Standard for Software Quality Assurance Plans," Computer, 12: 43-49 (August 1979).

13. Bunyard, Major General Jerry Max, USA and James Mike Coward. "Today's Risks in Software Development -- Can They Be Significantly Reduced?" Concepts, 5: 73-94 (Autumn 1982).

14. Burch, John G., Jr. and others. Information Systems: Theory and Practice (Third Edition). New York: John Wiley and Sons, Inc., 1983.

15. Campanizzi, J. A. "Structured Software Testing," Quality Progress, 17: 14-15 (May 1984).

16. Department of Defense. Technical Reviews and Audits for Systems, Equipment, and Computer Programs. MIL-STD-1521A. Washington: Government Printing Office, 21 December 1981.

17. -----. Software Quality Assurance Program Requirements. MIL-S-52779A. Washington: Government Printing Office, 1 August 1979.

18. -----. Quality Assurance Program. AFR 74-1. Washington: Government Printing Office, 1 June 1979.

19. -----. Configuration Practices for Systems, Equipment, Munitions, and Computer Programs. MIL-STD-483. Washington: Government Printing Office, 31 December 1970.

20. -----. Quality Assurance Terms and Definitions. MIL-STD-109B. Washington: Government Printing Office, 4 April 1969.

21. -----. Specifications Practices. MIL-STD-490. Washington: Government Printing Office, 30 October 1968.

22. DeRoze, Barry C. and Thomas H. Nyman. "The Software Life Cycle -- A Management and Technological Challenge in the Department of Defense," IEEE Transactions on Software Engineering, 4: 309-318 (July 1978).

23. Dobbins, James A. "Software Quality Assurance," Concepts, 5: 108-118 (Autumn 1982).

24. Drezner, S. M. and others. The Computer Resources Management Study. Report No. R-1855/1-PR. Rand, Santa Monica CA, April 1976.

25. Dunn, Robert and Richard Ullman. Quality Assurance for Computer Software. New York: McGraw-Hill, Inc., 1982.

26. Emory, C. William. Business Research Methods. Homewood IL: Richard D. Irwin, Inc., 1980.

27. Frank, Werner L. Critical Issues in Software. New York: John Wiley and Sons, Inc., 1983.

28. Gansler, J. S. "Keynote: Software Management," Computer Software Engineering, edited by Jerome Fox. Brooklyn NY: Polytechnic Press, 1976.

29. Glass, Robert L. Software Reliability Guidebook. Englewood Cliffs NJ: Prentice-Hall, Inc., 1979.

30. Grinath, Arthur C. and Phil H. Vess. "Making SQA Work: The Development of a Software Quality System," Quality Progress, 16: 18-23 (July 1983).

31. Grove, H. Mark. "DoD Policy for Acquisition of Embedded Computer Resources," Concepts, 5: 9-36 (Autumn 1982).

32. Hannan, James. "QA Needed for Effective Software," Government Computer News, 3: 29+ (April 1984).

33. Ingrassia, Frank S. "The Unit Development Folder (UDF): An Effective Management Tool for Software Development," Tutorial: Software Management. 249-262. IEEE, Inc., New York, 1979.

34. Institute of Electrical and Electronics Engineers. Standard Glossary of Software Engineering Terminology. IEEE Std 729. IEEE, Inc., New York, 1983.

35. Knight, Garney M. "Organizational Planning for Software Quality," Software Quality Management, edited by John D. Cooper and Matthew J. Fisher. New York: Petrocelli Books, Inc., 1979.

36. Klucas, Lieutenant Colonel Casper H., USAF and others. "Joint Service Software Policy and Standards," Concepts, 5: 191-201 (Autumn 1982).

37. Lewis, Robert O. "Software Verification and Validation (V&V)," Software Quality Management, edited by John D. Cooper and Matthew J. Fisher. New York: Petrocelli Books, Inc., 1979.

38. Lloyd, David K. and Myron Lipow. Reliability: Management, Methods and Mathematics (Second Edition). Redondo Beach CA: Published by the Authors, 1977.

39. Marciniak, Colonel John J., USAF. "A Perspective on Military Software Standardization Efforts," Second Software Engineering Standards Applications Workshop. 19-23. IEEE, Inc., New York, 1983.

40. McCall, James A. "An Introduction to Software Quality Metrics," Software Quality Management, edited by John D. Cooper and Matthew J. Fisher. New York: Petrocelli Books, Inc., 1979.

41. Poston, Robert M. "Software Quality Assurance Implementation," The IEEE Computer Society's Sixth International Computer Software and Applications Conference. 356-357. IEEE, Inc., New York, 1982.

42. Product Quality in the Operational Environment. Research report. HQ AFSC, Wright-Patterson AFB OH, November 1979.

43. Reifer, Donald J. "Software Quality Assurance Tools and Techniques," Software Quality Management, edited by John D. Cooper and Matthew J. Fisher. New York: Petrocelli Books, Inc., 1979.

44. -----. "A Glossary of Software Tools and Techniques," Computer, 10: 52-59 (July 1977).

45. Rubey, Raymond J. "The Effect of Standardization on Avionics Software Quality Assurance," IEEE National Aerospace and Electronics Conference. 656-662. IEEE, Inc., New York, 1979.

46. -----. "Planning for Software Reliability," Proceedings: Annual Reliability and Maintainability Symposium. 495-499. IEEE, Inc., New York, 1977.

47. Srinivasan, C. A. and Paul E. Dascher. "Quality Assurance Program: A Method to Improve Software Management," Hospital Financial Management, 35: 24-26+ (June 1981).

48. Stamm, Stephen L. "Assuring Quality Quality Assurance," Datamation, 27: 195-198+ (March 1981).

49. Systems Architects, Inc. Improving Software Quality Assurance Methods. Report No. RADC-TR-82-106. Rome Air Development Center, Griffiss AFB NY, April 1982 (AD-A116 980).

50. System Development Corporation. Software Acquisition Management Guidebook: Software Quality Assurance. Report No. ESD-TR-77-255. Electronic Systems Division, Hanscom AFB MA, August 1977 (AD-A047 318).

51. TRW Defense and Space Systems Group. Airborne Systems Software Acquisition Engineering Guidebook for Quality Assurance. Report No. ASD-TR-78-8. ASD/ENAI, Wright-Patterson AFB OH, November 1977 (AD-A059 068).

52. TRW Systems Group. Software Development and Configuration Management Manual. TRW Software Series No. TRW-SS-73-07. December 1973.

53. Walters, Gene F. "Application of Metrics to a Software Quality Management (AM) Program," Software Quality Management, edited by John D. Cooper and Matthew J. Fisher. New York: Petrocelli Books, Inc., 1979.

54. Webster's New Collegiate Dictionary. Springfield MA: G. and C. Merriam Co., 1979.

55. White, Benjamin B. "Planning for Software Quality," IEEE National Aerospace and Electronics Conference. 230-235. IEEE, Inc., New York, 1978.

56. Yeh, Raymond T. Current Trends in Programming Methodology. Englewood Cliffs NJ: Prentice-Hall, Inc., 1977.

VITA

Captain Steven P. Lamb was born on 26 August 1952 in Amarillo, Texas. He graduated from high school in Widefield, Colorado, in 1970 and attended Southern Colorado State College from which he received the degree of Bachelor of Science in Education in March 1975. In September 1979, he was commissioned in the Air Force through the Officer Training School program at Lackland Air Force Base, Texas. Following training, he completed the Computer Systems Development Officer Course at Keesler Air Force Base, Mississippi, in January 1980. He was then assigned as Chief, Communications System Segment Configuration Management Office, Headquarters North American Aerospace Defense Command, Cheyenne Mountain Complex, Colorado, until entering the School of Systems and Logistics, Air Force Institute of Technology, in June 1983.

Permanent address:

4825 Astrozon Blvd., Lot 191
Colorado Springs, Colorado 80916

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

## REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT<br>Approved for public release; distribution unlimited |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S)<br>AFIT/GSM/LSY/84S-19 | 5. MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a. NAME OF PERFORMING ORGANIZATION<br>School of Systems and Logistics<br>6b. OFFICE SYMBOL (If applicable)<br>AFIT/LS | 7a. NAME OF MONITORING ORGANIZATION |
| 6c. ADDRESS (City, State and ZIP Code)<br>Air Force Institute of Technology<br>Wright-Patterson AFB, Ohio 45433 | 7b. ADDRESS (City, State and ZIP Code) |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION<br>8b. OFFICE SYMBOL (If applicable) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER |
| 8c. ADDRESS (City, State and ZIP Code) | 10. SOURCE OF FUNDING NOS.<br>PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO. |
| 11. TITLE (Include Security Classification)<br>See Box 19 | |
| 12. PERSONAL AUTHOR(S)<br>Steven P. Lamb, B.S., Captain, USAF | |
| 13a. TYPE OF REPORT<br>MS Thesis<br>13b. TIME COVERED<br>FROM _____ TO _____ | 14. DATE OF REPORT (Yr., Mo., Day)<br>1984 September<br>15. PAGE COUNT<br>125 |
| 16. SUPPLEMENTARY NOTATION<br>LYNN E. WOLAVER<br>Dean for Research and Professional Development<br>14 Sept 84 | |
| 17. COSATI CODES<br>FIELD: 09 / GROUP: 02 / SUB. GR. | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)<br>Quality Assurance, Quality Control, Software Quality, Software Reliability, Software Management |

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: A SURVEY AND EVALUATION OF SOFTWARE QUALITY ASSURANCE

Thesis Advisor: Ronald H. Rasch, Major, USAF

| | |
|---|---|
| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT<br>UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐ | 21. ABSTRACT SECURITY CLASSIFICATION<br>UNCLASSIFIED |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL<br>Ronald H. Rasch, Major, USAF | 22b. TELEPHONE NUMBER (Include Area Code)<br>513-255-4020<br>22c. OFFICE SYMBOL<br>AFIT/LSB |

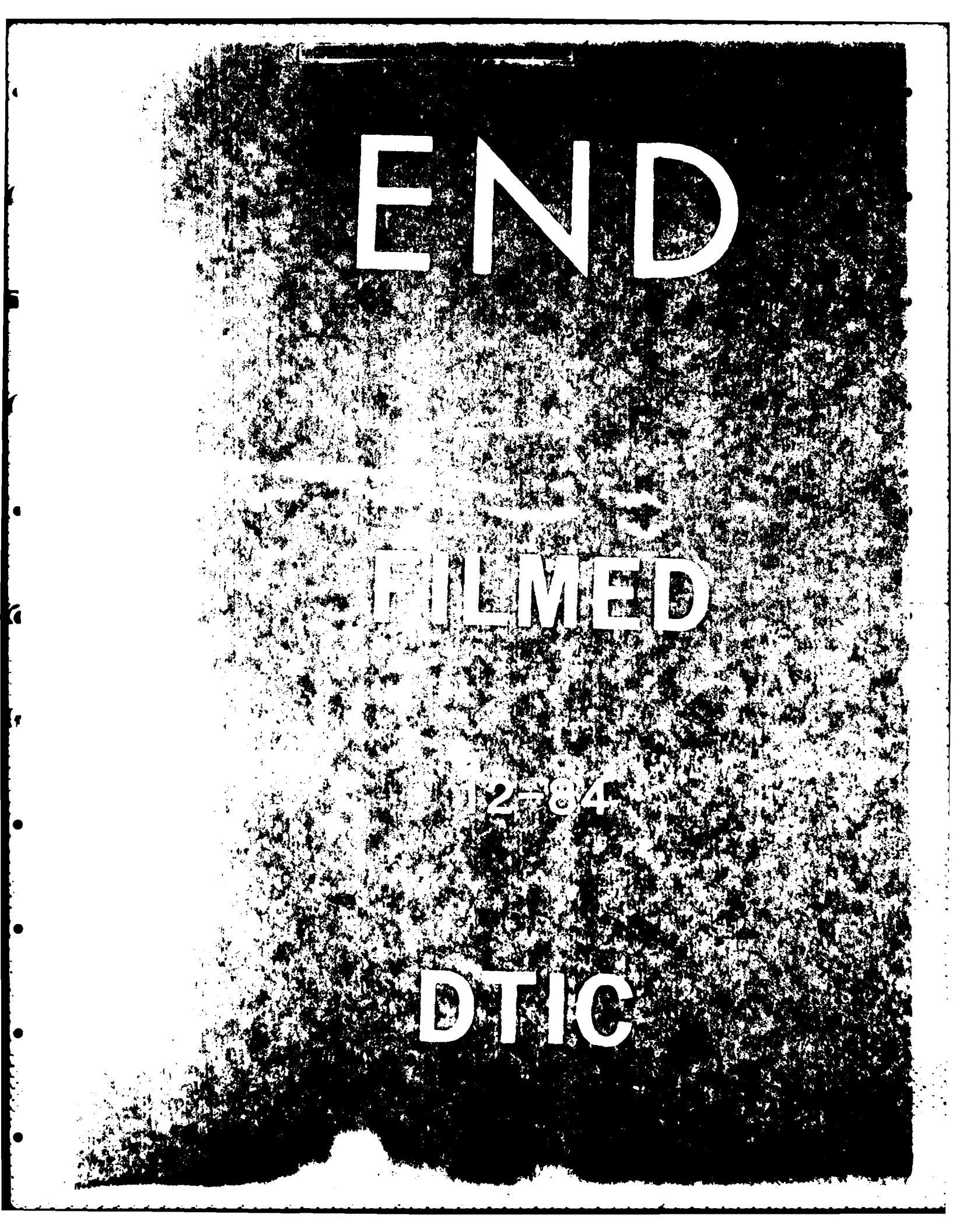
DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

It is crucial to the success of mission critical computer resources (MCCR) that software be delivered for operational use with the minimum number of errors possible. For this reason, the discipline of software quality assurance is needed. This research focuses on data collection by means of an extensive literature review and personal interviews with civilian and Air Force software development organizations. Then analysis was performed to determine what approaches would improve the quality of software before delivery to the Air Force for operational use.

To provide the highest level of software quality, the entire development process must include quality checks at each step from design through acceptance test. An active software quality assurance program that identifies and corrects errors during the development process is necessary. This effort will lead to significant defects being identified and resolved early. If the quality of software is to improve, greater emphasis must be placed on software quality assurance as a separate discipline. Quality software cannot be attained by following hardware oriented plans and procedures. Therefore, software conformance standards must be provided. Technology is constantly changing and advancing, and provision must be made to update personnel in the state-of-the-art quality assurance practices. Continual training is essential, both for those personnel who have quality assurance background and those who do not. The arguments for software quality assurance are critical. In short, they are to combat error and improve software quality to meet mission needs.


END
FILMED
12-84
DTIC